Holistic Processing of Hierarchical Structures in Connectionist Networks

Jane Neumann

PhD
The University of Edinburgh

2001

In memory of

Thilo Penzl

a brilliant mathematician,

a life-long friend

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Edinburgh, 1st August 2001 Jane Neumann

Acknowledgements

"One of the great joys of science lies in the moment of shared discovery."

James L. McClelland
David E. Rumelhart

I was given the chance to work for four years in a truly interdisciplinary, inspiring, and creative environment. Without the support of all the people who created this unique atmosphere, this work would not have been possible.
My foremost thanks go to my supervisors Mark Steedman and David Willshaw. While they gave me the freedom to follow my own ideas, they had an open ear whenever I needed it and I am grateful for many interesting discussions during the course of my work and for helpful comments on earlier drafts of this thesis.
I wish to thank Mark Ellison for supervision during the first year of my work, the University of Edinburgh for offering excellent facilities, and Betty Hughes who so often helped in administrative matters which, even to a German, sometimes seemed so inscrutable.
I am deeply indebted to Tony Plate, Lars Niklasson, William Self, and Kay Estler for many helpful suggestions and insights into the depths of mathematics. A special thank goes to Gert Westermann who helped with the first implementation of the GNG network.
This work was financially supported by the Gottlieb Daimler- und Karl Benz Stiftung, Germany, the EPSRC, the Institute for Communicating and Collaborative Systems and the Informatics Graduate School at the University of Edinburgh.
Special thanks go to my office mates and fellow students for offering many inspiring serious and not so serious discussions, an encouraging sense of optimism and solidarity, and uncountable numbers of doughnuts and cups of coffee. Yes, Nicola, you do make the best coffee in the world!
I want to thank Kay, Viv, Pete, Carolin, Becky, Kate, Suzanne, Kankel, and all the climbing and walking enthusiasts who dragged me away from my desk to the mountains and the beautiful coasts of Scotland and shared with me some of the most memorable days of these four years. Without Ian, Stan, Ollie and all my friends who so often reminded me that life extends beyond my desk and the topic of my PhD, I would not have been able to find the energy and enthusiasm to push through the unavoidable difficulties in the life of every PhD student. Paul, thanks for kicking and for reminding me of what used to be so important to me just when I was almost about to forget.

Mein größter Dank jedoch gilt meinen Eltern, die mich zu einem neugierigen, wissensdurstigen Menschen erzogen haben und ohne deren Liebe, Vertrauen und Toleranz diese Arbeit nicht möglich gewesen wäre.

Abstract

Despite the success of connectionist systems to model some aspects of cognition, critics argue that the lack of symbol processing makes them inadequate for modelling high-level cognitive tasks which require the representation and processing of hierarchical structures. In this thesis we investigate four mechanisms for encoding hierarchical structures in distributed representations that are suitable for processing in connectionist systems: Tensor Product Representation, Recursive Auto-Associative Memory (RAAM), Holographic Reduced Representation (HRR), and Binary Spatter Code (BSC). In these four schemes representations of hierarchical structures are either learned in a connectionist network or constructed by means of various mathematical operations from binary or real-value vectors.

It is argued that the resulting representations carry structural information without being themselves syntactically structured. The structural information about a represented object is encoded in the position of its representation in a high-dimensional representational space. We use Principal Component Analysis and constructivist networks to show that well-separated clusters consisting of representations for structurally similar hierarchical objects are formed in the representational spaces of RAAMs and HRRs.

The spatial structure of HRRs and RAAM representations supports the holistic yet structure-sensitive processing of them. Holistic operations on RAAM representations can be learned by backpropagation networks. However, holistic operators over HRRs, Tensor Products, and BSCs have to be constructed by hand, which is not a desirable situation. We propose two new algorithms for learning holistic transformations of HRRs from examples. These algorithms are able to generalise the acquired knowledge to hierarchical objects of higher complexity than the training examples. Such generalisations exhibit systematicity of a degree which, to our best knowledge, has not yet been achieved by any other comparable learning method.

Finally, we outline how a number of holistic transformations can be learned in parallel and applied to representations of structurally different objects. The ability to distinguish and perform a number of different structure-sensitive operations is one step towards a connectionist architecture that is capable of modelling complex high-level cognitive tasks such as natural language processing and logical inference.

Contents

1 Introduction **1**

1.1 Connectionism, Productivity, and Systematicity 6

1.2 Aim of the Thesis 11

1.3 Outline of the Thesis 12

2 Representation of Hierarchical Structures in Connectionist Networks **15**

2.1 Connectionist Representation 15

2.1.1 Connectionist Networks 15

2.1.2 Local and Distributed Representation 17

2.1.3 Similarity of Distributed Representations 19

2.1.4 The Problem of Representing Structured Objects 20

2.1.5 Reduced Representation 26

2.2 Tensor Product Representation 30

2.3 Recursive Auto-Associative Memory 32

2.4 Holographic Reduced Representation 35

2.5 Binary Spatter Code 42

2.6 Summary 43

3 Holistic Operations on Connectionist Representations of Structures **45**

3.1 Two Definitions 46

3.2 Holistic Operations on RAAM Structures 47

3.2.1 Classification of Structures 47

3.2.2 Matching of Structures 51

3.2.3 Transformation of Structures 53

3.2.4 Discussion 62

3.3 Holistic Operations on HRRs, Tensor Products, and BSCs 65

3.3.1 Matching of HRR structures 65

3.3.2 Transformations of Tensor Products, HRRs, and BSCs 67

3.3.3 Discussion 73

3.4 Summary 76

4 Learning the Holistic Transformation of HRRs **79**
4.1 Transformation of a Single Structure 80
4.2 Transformation of a Class of Structures 80
4.2.1 Learning by Gradient Descent 81
4.2.2 One-shot Learning 87
4.3 Systematicity and Generalisation 90
4.4 Discussion 94
4.5 Summary 98

5 The Spatial Structure of HRRs and RAAM Representations **101**
5.1 Constituent and Spatial Structure of Representations 101
5.2 Clustering 104
5.2.1 Hierarchical and Non-Hierarchical Clustering 104
5.2.2 Probability Density Estimation and EM 106
5.2.3 Connectionist Networks for Clustering 108
5.2.4 Growing Neural Gas 110
5.2.5 Principal Component Analysis 113
5.3 Spatial Structure of RAAM Representations 114
5.3.1 Previous Results 114
5.3.2 Our Own Experiments 117
5.4 Spatial Structure of HRR 122
5.4.1 Clustering of Transformation Inputs 123
5.4.2 Generalisation to Novel Elements 126
5.4.3 Generalisation to Structures of Higher Complexity 128
5.5 Summary 132

6 Parallel Transformation of Different Classes of HRR Structures **135**
6.1 The Problem of Parallel Transformations 136
6.1.1 Learning the Transformation Vector 138
6.1.2 Constructing the Transformation Vector 144
6.2 A Solution 149
6.2.1 Learning the Transformation Vector 151
6.2.2 Constructing the Transformation Vector 154
6.3 Summary 158

7 Conclusion **161**

List of Figures

2.1 A three-layer feedforward connectionist network 16
2.2 Full and reduced representation of a hierarchical structure 28
2.3 A Tensor Product 31
2.4 A dual RAAM 33
2.5 A recursive structure represented as a right branching tree 33
2.6 A tree learned by a ternary RAAM with additional memory 35
2.7 Circular convolution of two vectors with three elements 37

3.1 Five different tree structures 49
3.2 A dual labelling RAAM 49
3.3 Examples of the tree matching operation 52
3.4 Syntax trees for active and passive sentences 67
3.5 Transformation of syntax trees represented by tensor products 69

4.1 Mean square error when learning the transformation of a single structure using gradient descent 84
4.2 Mean square error when learning the transformation of a class of structures using gradient descent 86

5.1 Hierarchical and non-hierarchical clustering 105
5.2 A self-organising map showing the structure of the input data in the representational space 110
5.3 A GNG network showing the structure of the input data in the representational space 112
5.4 Tree structures encoded by a dual RAAM 117
5.5 First and second principal component of four classes of structures represented by a ternary RAAM 118
5.6 First and second principal component of four classes of structures obtained from a fully trained TN 119
5.7 First and second principal component of four classes of structures obtained from a TN trained without structures containing the element s . 120
5.8 First and fourth principal component of a single class of structures . . 121
5.9 The difference between a RAAM and HRR structure 125

5.10 Distances between the cluster centroids and data points representing HRR structures 126
5.11 Distances between the cluster centroids and data points representing HRR structures (2) 128
5.12 Distances between the cluster centroids and data points representing HRR structures (3) 130
5.13 Distances between the cluster centroids and data points representing HRR structures (4) 132

6.1 Distances between the cluster centroids and data points representing input structures for 4 different transformations. 143
6.2 Distances between the cluster centroids and data points representing input structures with class types for 4 different transformations. 153

Chapter 1

Introduction

What makes connectionist architectures attractive as models of cognition is that they exhibit properties not generally provided by traditional symbolic systems but found in human cognition, such as graceful degradation, the ability to handle noise and uncertainty, massively parallel processing, and the ability to learn from examples. These properties seem particularly well suited to modelling so-called low-level cognitive processes such as pattern recognition, pattern completion, and categorisation, which are found, for example, in visual perception and memory.

The early success of Connectionism in these areas of cognition led connectionist researchers to believe that neural networks can also account for high-level cognitive tasks such as natural language processing and reasoning and thus provide an alternative to traditional symbolic systems as explanations of cognition. One of the best known and most discussed examples of early connectionist implementations of higher cognitive processes is Rumelhart and McClelland's (1986) network which learns to form the past tense of English verbs. Other models aimed particularly at problems in natural language processing include a connectionist network that learns the pronunciation of English words from written text (Sejnowski and Rosenberg, 1987) and networks for sentence processing and children's language acquisition (McClelland and Kawamoto, 1986; St. John and McClelland, 1989; Elman, 1991; Plunkett and Marchman, 1991).

Such models, ascribing to what is often called *strong* Connectionism, are based on the assumption that the traditional symbolic data structures and the language-like rules of symbol manipulation systems can be discarded in favour of representations and

behaviour that emerge from learning in a homogeneous net of simple processing units. Strong Connectionism seeks to show that the similarity-based representations formed in neural networks and their learned mappings eliminate the necessity to represent symbols, variables, and rules explicitly. The generalisation of acquired knowledge to new objects in such networks is based exclusively on the similarity of new input patterns to already existing representations.

Models based on these assumptions were, and still are, subject to heavy criticism, in particular on the grounds of their lack of both generative capacity and task control and their inability to handle compositional and systematic aspects of cognition (Fodor and Pylyshyn, 1988; Pinker and Prince, 1988; Marcus et al., 1995; Fodor, 1999). Most of the criticisms come from advocates of the traditional symbolic school, which assumes as vehicles of cognition symbol manipulation systems inspired by Turing machines and the von Neumann architecture of conventional computers. This view is concisely expressed in Newell and Simon's (1976) Physical Symbol Systems Hypothesis, which states that a physical symbol system has the necessary and sufficient means for general intelligent action. According to Harnad (1990), a physical symbol system comprises a set of arbitrary physical tokens, the symbols, which are manipulated on the basis of explicit rules. Symbol manipulation processes "consist of 'rulefully combining' and recombining symbol tokens" and are based purely on the shape of the symbols (Harnad, 1990). The arbitrariness of the symbol shapes and the explicitness of the rules are in stark contrast to connectionist networks, where representations develop during learning depending on the similarity of the objects they represent and the context they appear in and where operations over these representations are comprised of learned mappings. Moreover, classical symbol systems make an explicit distinction between categories and tokens and implement variables and processes for labelling tokens and instantiating variables. Generalisations to new objects in these systems are expressed in terms of relations between categories in a domain rather than in terms of their similarity to already existing objects (Marcus, 1998).

Such systems, Symbolists argue, provide a natural account of several aspects of cognition such as the distinction between atomic and complex objects, the ability to generalise knowledge to new members of a category, and the ability to link elements in a discourse (Marcus, 1998). However, the proposal that the mind essentially im-

plements a symbol processing machine is still as much to be proven as the strong Connectionists' claim that a network of neuron-like units which does not simply implement a symbol processor can fully account for cognition. While no connectionist system is yet able to model all aspects of the cognitive task it implements, symbolic systems such as Turing machines, although theoretically very powerful, provide only unsatisfactory explanations of cognition in general and the acquisition of cognitive knowledge in particular.

The struggle on both sides to provide good explanations of the phenomena found in human cognition has brought Symbolists and Connectionists closer together in recent years. Symbolists, for example, acknowledge that the serial processing in most symbolic systems is in stark contrast to the brain operating largely in parallel. The contribution of connectionist networks to a better understanding of cognition could therefore at least lie in the specification of how symbolic operations are performed in parallel (Marcus, 1998). This so-called implementationalist view is also shared by a large number of Connectionists, such as Dyer (1991) who points out:

> "If we could embed symbol representations and structure-manipulating operations within a distributed, sub-symbolic architecture, then very powerful, massively parallel, fault tolerant high-level reasoning/planning systems could be created."

Others argue that a large space of possible integrations of Symbolism and Connectionism can be opened up by relaxing the strong assumptions of the two paradigms. Hadley (1999), for example, maintains that one direction of research into such integration is to replace the homogeneous network assumed by strong Connectionism by a modular architecture. While this architecture could be 'classical' with respect to task control and the information exchange between modules, on the level of sub-modules it could be purely connectionist. Touretzky and Pomerleau (1994) observe that when relaxing the Symbolists' assumption that symbols are of arbitrary shape, similarity-based connectionist representations functioning as names for objects could practically play the roles of symbols. The properties of such representations could then be exploited by symbolic and non-symbolic processes alike.

Such theoretical considerations are supported by a large amount of empirical studies into closing the gap between the two paradigms. Miikkulainen and Dyer (1991),

for example, proposed a hierarchically organised modular connectionist network as the basis for a natural language processing system. Sun's (1995) connectionist system for robust reasoning is based on the combination of local and distributed representations of objects. The former support symbolic rule-based reasoning while the latter allow sub-symbolic similarity-based generalisations over the same objects.

The attempt to treat Symbolism and Connectionism as two possible approaches to cognition, one not to the exclusion of the other, has led to the development of hybrid architectures (see, e.g., Sun and Alexandre (1997); Wermter and Sun (2000)). Both connectionist networks and symbol processors have properties which are desirable in a model of cognition. The noise tolerance, the ability to learn from examples, and the similarity-based representation in connectionist systems and the expressive power of symbolic systems are only a few examples. Hybrid architectures containing connectionist and symbolic components are based on the idea that the combination of the best properties of both concepts can overcome the difficulties observed for symbolic and connectionist systems on their own, such as the brittleness of the former and the lack of explanatory power of the latter.

Other research has gone into making the mappings learned and the information stored in connectionist networks accessible and transparent by extracting symbolic rules that describe the behaviour of a network in easily understandable form (Andrews et al., 1995; Neumann, 1996). Rules extracted from connectionist networks can provide the interface between the connectionist and symbolic parts in hybrid systems and offer explanations of what a network has learned. The ability to explain a network's behaviour is not only important, for example, for the commercial application of connectionist networks in safety-critical expert systems (Gallant, 1988), it also provides some understanding of the computational power of connectionist networks in general and their relations to other computational paradigms. For instance, Giles and Omlin (1993) showed that simple recurrent networks (Jordan, 1989; Elman, 1991) implement Finite State Automata, which in turn have wide applications in natural language processing (see, e.g., Roche and Schabes (1997)). Moreover, the successful extraction of symbolic rules from connectionist networks suggests that the information which symbolic and connectionist architectures are able to represent might not be as different as is often assumed, although it is acquired and processed in different ways. Whereas

symbol manipulation rules are explicitly provided in symbolic systems, the information extracted from a connectionist network describes the mapping between input and output representations which is usually learned from examples and encoded in the network architecture and the weighted connections between processing units. However, the ability to extract this information in the form of symbolic rules supports the hypothesis that the difference between symbolic systems and connectionist networks might not lie in the presence or absence of rules as proposed by both strong Symbolists and Connectionists, but only in the nature of the rules symbolic and connectionist systems contain (Elman, 1989).

One of the most important points that Symbolists and Connectionists agree on today is the need for structure in the representations as well as the architectures underlying cognition. As Levy and Pollack (2001) point out, one possible approach to connectionist modelling of high-level cognition is to acknowledge "the need for systematic, compositional structure, but [to] reject traditional, exceptionless linguistic rules in favor of the flexible computation afforded by connectionist representations." Adopting this approach allows Connectionists to increase the representational power of neural networks while retaining their ability to learn the processing of these representations from examples. This is the approach to connectionist modelling taken in this thesis. As Levy and Pollack (2001) further observe, modelling cognition under these assumptions requires solving two problems. Firstly, we need to develop connectionist representations for compositional structured objects. Secondly, we have to show how these connectionist representations can support the kind of processes traditionally viewed as mediated by rules.

While compositional structured objects and variable bindings can be straightforwardly represented in symbolic systems using symbols for atomic constituents and concatenations of symbols for complex objects, connectionist architectures do not provide a mechanism per se for representing structured objects. However, in recent years, progress has been made regarding the connectionist representation of compositional objects, variables, and dynamic bindings. Touretzky and Hinton (1988) proposed a connectionist production system where sets of triples are stored in a coarse-coded distributed memory. Shastri and Ajjanagadde (1993) used temporal synchrony to represent variable bindings in a localist connectionist model of reflexive reasoning. Smolen-

sky (1990) proposed the use of Tensor Products for the representation of variable bindings and symbolic structures in connectionist networks.

Influential theoretical background for the connectionist representation of structure has been provided with the notions of *functional compositionality* (van Gelder, 1990) and *reduced representation*[1] (Hinton, 1990). Representational schemes implementing these concepts include Recursive Auto-Associative Memory (RAAM) (Pollack, 1988), Holographic Reduced Representation (HRR) (Plate, 1994), and Binary Spatter Code (BSC) (Kanerva, 1997). These representational schemes, together with Smolensky's Tensor Product Representation, form the basis for the work presented in this thesis.

Our own research concentrates on the second task outlined above, the development of connectionist processes traditionally viewed as rule-like, over connectionist representations of structured objects. It has recently been argued that connectionist representations of hierarchical objects support not only structure-sensitive but *holistic* processes, i.e., processes that operate directly on the representations of composed objects and do not require their decomposition into constituents (Plate, 1994; Niklasson and Bodén, 1997; Kanerva, 1998). In this thesis we investigate how such processes can be learned from examples and show that they can overcome two main criticisms of connectionist architectures: their lack of systematicity and productivity.

1.1 Connectionism, Productivity, and Systematicity

Some of the strongest objections to Connectionism come from Fodor and colleagues (Fodor, 1999; Fodor and McLaughlin, 1990; Fodor and Pylyshyn, 1988). Like other Symbolists, Fodor views cognitive processing as strictly symbolic. He proposes a Language of Thought (Fodor, 1975) which assumes

> "... mental representations to have *a combinatorial syntax and semantics*, in which (a) there is a distinction between structurally atomic and structurally molecular representations; (b) structurally molecular representations have syntactic constituents that are themselves either structurally

[1] The concepts of functional compositionality and reduced representation are defined and discussed in detail in Chapter 3.

> molecular or structurally atomic; and (c) the semantic content of a (molecular) representation is a function of the semantic contents of its syntactic parts, together with its constituent structure." (Fodor and Pylyshyn, 1988, p.12, original emphasis)

Starting from this proposal, Fodor and colleagues then argue that, although Connectionism agrees that mental states are representational and the objects to be represented are structured, "it acknowledges neither syntactic nor semantic structure in mental representations" (Fodor and Pylyshyn, 1988, p.49) and thus cannot provide an adequate explanation of cognition. While this argument is true for implementations based on the assumptions of strong Connectionism, it no longer holds for connectionist architectures that employ functional composition and reduced representations. Such connectionist systems are able to represent atomic constituents and to combine them to hierarchical objects. However, the resulting representations do not typically show the classical constituent structure found in symbolic systems, where concatenation is used as the mechanism for composing complex objects. Whereas most[2] symbolic representations of composed objects contain the full representations of their constituents, connectionist representations of the constituents of structured objects are often superimposed. This means that an object might be represented in different ways in the same system, depending on whether it is represented on its own or as part of a small or a larger structure[3].

By itself, the ability to represent structured objects does not explain cognition, however. According to Fodor, a second property of the mind which a cognitive model has to account for is the structure-sensitivity of mental processes. Fodor and Pylyshyn (1988, p.32) observe that "mental processes are characteristically sensitive to the combinatorial structure of the representations on which they operate." Moreover, such structure-sensitive operations

[2] An exception are symbolic pointers.

[3] We believe, however, that Fodor has been misinterpreted by a number of connectionist researchers with respect to the way molecular representations are formed (see, e.g., van Gelder (1990) and Niklasson and van Gelder (1994a)). Postulating a constituent structure of representations does not imply that concatenation is the only mechanism of constructing structured objects from constituents and, to our best knowledge, Fodor nowhere claims concatenation to be the mode of composition in the Language of Thought.

> "...can apply equally to representations that differ widely in their structural complexity. The operation that applies to representations of the form $P \& Q$ to produce P is satisfied by, for example, an expression like '$(A \vee B \vee C) \& (D \vee E \vee F)$', from which it derives the expression '$(A \vee B \vee C)$'." (Fodor and Pylyshyn, 1988, p.13)

For a system to implement such functionality it must be *productive* at least to the degree that it is able to represent expressions of variable size. It further has to be *systematic* in that the ability to apply an operation to one representation implies the ability to apply the same operation to a structurally related representation. Note that the need for systematic processing was already anticipated, although not met, in early connectionist implementations. Elman (1989), for example, when discussing his connectionist model for the acquisition of grammatical structure, points out that "one would like to know whether the network can inferentially extend what it knows about the types of noun phrases encountered ... to noun phrases with different structure."

Although we agree that operations over combinatorial structures are only truly useful if they show some degree of systematicity and productivity, we disagree with Fodor and other Symbolists that only systems implementing symbolic representations and operations in the traditional sense are capable of doing so. The connectionist representations and processes investigated in this thesis are productive and show a very high degree of systematicity.

The productivity of a system refers to its capacity to produce and represent new combinatorial objects, such as a new sentence in a natural language, beyond those already present in a corpus (Bechtel, 1999). This is often argued to be problematic for connectionist networks, given the usually finite resources available for the representation of objects: "Connectionist cognitive architectures cannot, by their very nature, support an expandable memory, so they cannot support productive cognitive capacities" (Fodor and Pylyshyn, 1988, p. 35). We do not agree with this argument because, we believe, it addresses the storage capacity of connectionist systems rather than their in-principle ability to generate novel structured objects. Moreover, the argumentation is flawed, as it directly compares connectionist networks having fixed resources with Turing machines and symbolic systems that have an unbounded tape or memory. If we allow, as Fodor and Pylyshyn do, symbolic systems to have an expandable memory

without increasing the representational resources, and if we allow humans to use pencil and paper in order to do their sums, then we also have to allow connectionist networks to have mechanisms to (externally) store and recall parts of structured objects.

It has been shown that connectionist architectures like Recursive Auto-Associative Memories and networks implementing Holographic Reduced Representation are able to represent structured objects of variable size (Pollack, 1988; Plate, 1994). In addition, if connectionist networks employing HRR are augmented with an external memory for components of objects, i.e. with 'pencil and paper', then structures of unlimited size can be processed as long as the size of the components is kept small (Plate, 1994). Such connectionist systems are therefore not susceptible to criticism on the grounds of lack of productivity. The holistic processes investigated in this thesis are productive to the extent that they produce previously unseen hierarchical structures as part of the generalisation from unknown inputs. Moreover, these newly formed structures can be of higher complexity, i.e. larger, than all objects previously encountered.

Systematicity in connectionist systems is usually defined in terms of their capacity to generalise knowledge acquired from training examples to previously unseen objects. Hadley (1992) observes that learning-based systematic behaviour comes in degrees depending on the complexity of the training and test examples used. He distinguishes three levels of systematicity: weak, quasi-, and strong systematicity. A connectionist architecture can only be called strongly systematic if, after training with simple expressions, the learning system can process novel and more complex expressions which contain constituents in positions they did not occupy in the training examples. While humans undoubtedly possess this form of systematicity, he argues, it has not been observed in artificial connectionist networks so far.

Niklasson and van Gelder (1994a) refined Hadley's concept of learning-based systematicity and propose the following five levels:

Level 0 No generalisation. The system only remembers the training examples.

Level 1 Generalisation to novel composed structures. The constituents of the test structures appear in all their syntactically allowed positions during training. The system only generalises to new combinations of constituents.

Level 2 Generalisation to novel positions of constituents. The training set contains

all constituents of the test structures but not all of them in all their syntactically allowed positions.

Level 3 Generalisation to novel constituents. Some constituents of the test structures do not appear in the training set.

Level 4 Generalisation to novel complexity. Some test structures are of higher complexity than the structures used for training.

Level 5 Generalisation to novel constituents in structures of higher complexity. Some test structures are of higher complexity than the structures used for training and contain constituents that did not appear in the training set.

Throughout this thesis we use these five levels of systematicity for the analysis of the generalisation capacity of particular connectionist systems. They allow for a more precise characterisation of systematic processing in learning-based architectures than Hadley's proposed three levels while still capturing the immense differences between generalisations even on neighbouring levels. Imagine, for example, five connectionist networks which are able to represent expressions in propositional logic and are all trained to perform the same task: to respond with P when given the expression $P \,\&\, Q$ (cf. Fodor and Pylyshyn's proposition on Page 8). Suppose further, for the sake of simplicity, that these networks each achieve a different one of the five levels of systematicity after training with a single example only. After successful training, the networks with level 0 and level 1 systematicity would not show any generalisation capacity at all[4]. The network with level 2 systematicity would only be able to make three generalisations. It would correctly respond with P when given the expression $P \,\&\, P$ and with Q when given the expressions $Q \,\&\, P$ and $Q \,\&\, Q$. However, the network with level 3 systematicity would be able to answer the input $A \,\&\, Q$ with A, the input $B \,\&\, P$ with B, the input $B \,\&\, C$ with B, and so on, despite the fact that it had not encountered A, B, and C in the training set at all. The difference between these two neighbouring levels is significant, with level 2 allowing for only three

[4]A level 1 generalisation is not possible in this particular example, because both elements are only allowed in a single syntactic position in the training set and cannot be combined to a new structure in the test set.

generalisations and level 3 allowing for a theoretically infinite number. Note that the network with level 5 systematicity would, as required by Fodor, be able to respond with $(A \vee B \vee C)$ when given the input $(A \vee B \vee C) \,\&\, (D \vee E \vee F)$.

What Hadley refers to as strong systematicity can, using these five levels, be described as systematicity of level 4 or higher. Although some connectionist architectures have been shown to exhibit level 3 systematicity (Niklasson, 1993), to our best knowledge no connectionist system has yet achieved higher levels of systematic processing.

1.2 Aim of the Thesis

The work presented in this thesis primarily addresses processes that can be applied to hierarchical objects represented in connectionist networks. The central aim of the thesis is to show that connectionist representational schemes based on the concept of reduced representation and on the functional composition of hierarchical structures can support structure-sensitive processes which show a degree of systematicity previously unseen in connectionist architectures. We support our argument by giving evidence for the following points:

- Connectionist representational schemes such as RAAM and HRR allow for the connectionist representation of hierarchical objects by a fixed number of connectionist processing units.

- The structural similarity of represented objects is reflected in the spatial structure of the HRR and the RAAM representations. Representations of structurally similar objects form clusters in the high-dimensional representational space. This spatial structure of the representations supports their holistic yet structure-sensitive processing.

- The holistic transformation of structures represented by HRRs can be learned from examples. After training, the learning algorithm is able to generalise the acquired knowledge to representations of structures which are more complex than the training examples and which contain novel constituents. This generalisation capacity corresponds to a degree of systematicity required in high-level

cognition such as natural language processing and logical inference which, to our best knowledge, has not been observed in connectionist architectures so far.

While we are able to present in this thesis a connectionist learning method based on HRR that shows systematicity of level 5, there are still problems to be solved on the way to a full connectionist model of high-level cognitive processes. For example, a connectionist inference system must not only be able to perform a single inference over expressions of different complexity, it has to acquire and implement a number of *different* inference rules and must facilitate mechanisms that decide which rule to apply given a set of input expressions and mechanisms that distinguish valid from invalid inferences. We have conducted some experiments addressing the first of these tasks, the acquisition of a number of different transformational inferences over structured objects represented by HRR. Although the results have to be regarded as preliminary, they outline a promising direction for further research.

1.3 Outline of the Thesis

The remainder of this thesis is organised as follows.

In Chapter 2 we introduce the basic mechanisms of connectionist networks and connectionist representations. We then investigate the problems any architecture has to address in order to represent hierarchical structured objects and compare local and distributed connectionist representation with respect to their suitability for this task. We introduce two different modes of composition, concatenation and functional composition, and discuss their appropriateness for the composition of hierarchical structures in connectionist networks. We will argue that the combination of functional composition with the concept of reduced representation provides the means for representing hierarchical structured objects in connectionist architectures with a fixed number of representational resources. In the remainder of this Chapter, four different schemes for the representation of hierarchical structures in connectionist architectures are discussed: Tensor Product Representation, Recursive Auto-Associative Memory, Holographic Reduced Representation, and Binary Spatter Code.

Chapter 3 focuses on holistic structure-sensitive operations over hierarchical structures represented in these schemes. We first discuss the holistic classification, match-

ing, and transformation of RAAM structures. These operations can be learned by simple feedforward backpropagation networks. Results presented in the literature are reviewed and compared with our own experiments. We argue that although structures represented in the hidden layer of RAAMs can in principle be processed holistically, relatively complex holistic operations require the modification of the representations to fit the task at hand. The second part of this Chapter investigates the holistic matching and transformation of Tensor Products, HRRs, and BSCs. These operations are not learned by connectionist networks but constructed from the representations of constituents of the hierarchical objects.

In Chapter 4 we present two new methods for learning the holistic transformation of HRRs from examples. These methods not only overcome the weaknesses of the construction methods, they also show the capacity for generalisations that require a very high level of systematicity. Our experiments show that a learned transformation vector is able to generalise the acquired knowledge to structures containing entirely novel constituents and to structures of higher complexity than the training examples. These generalisations require systematic processing of level 3 and level 4 or 5, respectively.

Structure-sensitive processes in symbolic systems are based on the manipulation of the constituent structure present in symbolic representations. In Chapter 5 we show that connectionist representations of hierarchical objects exhibit another form of structure. They structure the representational space such that representations for similar objects form clusters. After introducing different clustering techniques, we apply Principal Component Analysis and a constructivist connectionist network to HRRs and RAAM representations which either serve as inputs for holistic transformations or represent results of learned holistic transformations. We argue that it is the structure of the representational space of HRR and RAAM representations that provides the information necessary for their holistic structure-sensitive processing. We further show that holistic transformations of HRRs preserve the structure of the representational space, allowing for a number of holistic operations to be applied successively.

The results presented in Chapters 3 and 4 suggest that a transformation vector for the holistic transformation of HRRs can transform a number of structurally different hierarchical objects in parallel. This issue is further pursued in Chapter 6. Using the example of 13 different transformations, we investigate conditions on the representations

that allow for their parallel transformation and compare the methods of constructing and learning the transformation vector with respect to these conditions. We again employ the cluster analysis of the representational spaces of HRR structures to show that for our example transformations a simple modification of the initial encoding scheme ensures the accurate transformation of structurally different objects in parallel.

Chapter 7 summarises the main results of our work and outlines related and future work.

Chapter 2

Representation of Hierarchical Structures in Connectionist Networks

In order for connectionist networks to model cognitive processes they need to be able to represent compositional structured objects. However, a connectionist network does not by itself provide mechanisms for the representation of relations between constituents in compositional structures. One key issue in connectionist research is therefore the development of special mechanisms for the composition, representation, and decomposition of structured objects in connectionist architectures. In this Chapter we present four different schemes for the connectionist representation of hierarchical structures. We first introduce the basic principles of connectionist representation. We then outline the problems any mechanisms for the representation of hierarchical structured objects has to address and discuss solutions within the connectionist framework.

2.1 Connectionist Representation

2.1.1 Connectionist Networks

A connectionist network usually consists of a fixed number of simple processing units, often also called neurons, which are connected by weighted links. These links are the only means of communication between the units. A unit receives a signal from one or more other units, calculates its activation according to a relatively simple activation

function, and sends the result to one or more other units.

The development of connectionist networks goes back to the early 1940's when McCulloch and Pitts (1943) developed the first artificial neuron. A few years later, Hebb (1949) proposed a learning mechanism that explained the classical psychological conditioning in animals as an effect of the properties of individual neurons. The first layered network of neurons, the *perceptron*, was developed by Rosenblatt (1958) and was applied to problems of pattern recognition. Despite this early enthusiasm, connectionist network research fell into a rapid decline in the late 1960's largely on the account of Minsky and Papert's (1969) publication of some fundamental limitations of perceptrons. They argued, for example, that a perceptron cannot even implement simple logical functions such as XOR. Moreover, they expected multi-layer networks to be subject to the same restrictions as the perceptron. However, with the discovery of new learning algorithms for multi-layer networks and the development of new network architectures the field advanced again in the 1980's.

Today, artificial neural networks come in a large variety of forms. Figure 2.1 shows an example of a simple three-layer feedforward network. The units are grouped according to their function into input layer, hidden layer, and output layer and neighbouring layers are fully connected, i.e., a unit receives signals from all units in the layer below and sends signals to all units in the layer above. Typical activation functions for

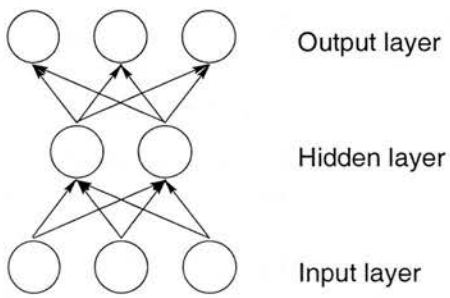

Figure 2.1: A tree-layer feedforward connectionist network with three input units, one layer of two hidden units, and three output units.

units in such a network are the step function and the logistic function. Other widely used network architectures include recurrent networks (Jordan, 1989; Elman, 1991), Hopfield networks (Hopfield, 1982), and Kohonen Feature Maps (Kohonen, 1982).

What all connectionist networks have in common is their ability to learn from ex-

amples. Learning is carried out by adjusting the weights of the connections between the units of the network. The weights are modified until the network behaves according to a specific task when presented with a number of training examples. Various methods for adjusting the weights are used depending on the network architecture and the task to be learned. A well-known and extensively studied example of a learning algorithm is the Backpropagation algorithm (Rumelhart et al., 1986; LeCun, 1985) for multi-layer feedforward networks. In this algorithm gradient descent is used to find the combination of weights in the network that best fits the task to be learned. Other training algorithms include Hebbian learning (Hebb, 1949) and self-organisation (Willshaw and von der Malsburg, 1976; Kohonen, 1982).

A prerequisite for successful learning in a connectionist network is the adequate representation of all information relevant to the task at hand. The only representational resources available are the units of the network, the connections between the units, and the network topology. Whereas the particular network architecture chosen and the weighted connections between units represent relations between objects and general domain knowledge (Plate, 1994), the objects in a domain are represented by patterns of activity over a number of units. The activations of the input units represent the input information to the network. The result of the network's calculation is represented by the activations of the output units. Hidden units are used to store intermediate results. From a more mathematical perspective the pattern of activity over n units in a network can be viewed as a n-dimensional vector of binary or real-valued elements.

2.1.2 Local and Distributed Representation

According to the way the units in a network are applied to the representation of objects, we can distinguish between local and distributed representations.

In a local representation each object of a domain is represented by a single unit. The activation of a particular unit refers to the presence of a particular object. The immediate advantage of such a representation is that it is unambiguous and can be easily interpreted and understood by humans. This makes local representation an obvious choice for the representation of inputs and outputs of connectionist networks. However, it is also a very inefficient and inflexible way of representing information. Local representation requires at least as many units as objects to be represented which pro-

hibits the representation of large domains. Further, increasing the number of objects in a domain is problematic given that a network architecture and with it the number of units representing objects is usually fixed[1].

In a distributed representation each object is represented by a pattern of activity distributed over many units and each unit is involved in representing many different objects (Hinton et al., 1986). Although such patterns of activity are much harder to interpret than the activation of a single unit, distributed representation has various advantages over local representation. Firstly, the representational resources are used very efficiently. The number of possible patterns of activity over a set of units, i.e., the number of objects that can be represented, is far greater than the number of units. Secondly, the similarity of objects can be expressed by the similarity of the patterns representing them. This allows for a fast comparison of representations and supports the ability of connectionist networks to generalise the knowledge acquired during learning to unknown objects. Finally, the large amount of redundancy usually present in distributed representation means that representations degrade gracefully when noise is added. This redundancy further allows for error correction and pattern completion in connectionist networks.

A distributed representation is usually found in hidden layers of connectionist networks where it develops during training. It is difficult to obtain distributed representations for objects that serve as inputs or targets for networks, however. Here, the patterns of activity over units are usually either hand-designed or randomly generated. While the first method of obtaining distributed representations can be very laborious, the latter no longer guarantees the similarity of representations for similar objects. A promising exception is the use of Latent Semantic Analysis (LSA) (Landauer and Dumais, 1997) for the acquisition of distributed representations for word meanings. LSA statistically analyses the co-occurrences of words in text documents, hereby deriving high-dimensional vector representations of words that reflect their semantic similarity. Such vectors, we believe, could potentially function as input vectors for connectionist architectures.

It should be noted that the boundary between local and distributed representation

[1] In constructivist networks the number of units and connections changes during learning.

is a vague one. Looking, for example, at the following representations of two objects

$$\begin{aligned} r_1 &= 111000 \\ r_2 &= 000111 \end{aligned}$$

some might argue that they are distributed, because more than one unit is involved in the representation of one object. On the other hand, the representations could be regarded as local, because each unit only contributes to the representation of exactly one object and the representations are not overlapping but completely distinct. Although the representations are distributed over a number of units, they still possess some characteristics of local representation. Such representations are therefore often referred to as *localist* representation.

2.1.3 Similarity of Distributed Representations

We will see throughout this thesis that the similarity of distributed representations plays a key role in the connectionist representation and processing of hierarchical structures. We therefore need suitable measures for the similarity of distributed representations. A commonly used measure can be found by viewing representations of objects distributed over n units as n-dimensional vectors that define points in the n-dimensional Euclidean space $\mathbb{R}^n$. The similarity of such points or vectors can be understood in terms of their distance in $\mathbb{R}^n$ measured by the *Euclidean distance*. The Euclidean distance between two n-dimensional vectors $\mathbf{X} = (x_1, \ldots, x_n)$ and $\mathbf{Y} = (y_1, \ldots, y_n)$ is defined as

$$d(\mathbf{X}, \mathbf{Y}) = \|\mathbf{X} - \mathbf{Y}\| \tag{2.1}$$

$$= \left(\sum_{k=1}^{n} (x_k - y_k)^2 \right)^{1/2} . \tag{2.2}$$

The similarity of these vectors is then expressed by the reciprocal Euclidean distance. The smaller the Euclidean distance between the vectors, i.e., the smaller the distances between the corresponding vector elements, the greater is the similarity between the vectors (Haykin, 1999). The Euclidean distance between two identical vectors is 0.

A second measure for the similarity of vectors is the normalised *inner product*, often also called *dot product*, of two vectors. The dot product of two n-dimensional

vectors $\mathbf{X} = (x_1, \ldots, x_n)$ and $\mathbf{Y} = (y_1, \ldots, y_n)$ is defined as

$$\mathbf{X} \cdot \mathbf{Y} = \mathbf{X}^T \mathbf{Y} \tag{2.3}$$

$$= \sum_{k=1}^{n} x_k \, y_k \, . \tag{2.4}$$

In order for the dot product to measure the similarity of two vectors it has to be normalised. Without normalisation it reflects the lengths of the vectors as much as their similarity. The normalised dot product, i.e., $\mathbf{X} \cdot \mathbf{Y}$ divided by $\|\mathbf{X}\|$ and $\|\mathbf{Y}\|$, is the cosine of the angle subtended between the vectors $\mathbf{X}$ and $\mathbf{Y}$. Thus, the normalised dot product of two identical vectors is 1. Note that the Euclidean distance and the normalised dot product are closely related by

$$d^2(\mathbf{X}, \mathbf{Y}) = 2 - 2\,(\mathbf{X} \cdot \mathbf{Y}) \tag{2.5}$$

for normalised vectors $\mathbf{X}$ and $\mathbf{Y}$. As this equation shows, minimising the Euclidean distance corresponds to maximising the dot product, i.e., the similarity between the vectors $\mathbf{X}$ and $\mathbf{Y}$.

2.1.4 The Problem of Representing Structured Objects

Representational Adequacy

In order for connectionist networks to successfully tackle complex high-level connectionist tasks, they have to employ not just any form of connectionist representation but adequate representations of all entities involved. Connectionist representations in general are often regarded as interesting alternatives to traditional symbolic representations because of their many advantageous computational properties such as

- Efficiency
- Immediate accessibility
- Noise tolerance
- Similarity of representations supporting generalisation.

However, as outlined in Chapter 1, the processing of hierarchical structures in a connectionist network in particular demands some additional properties that adequate representations of the objects involved should possess (cf. Plate, 1994):

- Accessibility of constituents: The connectionist network must be able to access not only representations of hierarchical structures as a whole but also representations of parts of these structures and the basic constituents the structures are composed of.
- Productivity: The mechanisms involved in the composition and decomposition of structures must support the composition of representations for previously unknown hierarchical structures from known and unknown constituents.
- Structure sensitivity: The representations of hierarchical objects should support their structure-sensitive systematic processing.

Despite their many advantages neither local nor distributed connectionist representation readily allow for the adequate representation of hierarchical structures. A closer look at the nature of hierarchical structured objects reveals some of the problems a method for the representation of structured objects has to address.

The Binding Problem

Hierarchical structured objects are composed of smaller objects by a recursive process. We will for the remainder of this thesis refer to atomic objects in a domain as *elements*. In order to form a hierarchical structure, elements are first combined into small structures. These can in turn be combined with other structures or more elements to form more complex hierarchical structures. A triangle in a plane, for example, can be interpreted as a structured object consisting of three smaller structures, the three corner points of the triangle, which in turn consist of two elements each, the x- and the y-coordinate. In order to represent the triangle as a whole, we need to be able to represent some information about all three corner points of the triangle. Representing a single point in a plane is simple. The simultaneous representation of the x-coordinate and the y-coordinate of the point tells us its exact position. However, if we want to represent two points, simultaneously activating the representations for the two x-coordinates and

the two y-coordinates is not sufficient, because we cannot decide whether the points are in the positions (x_1, y_1) and (x_2, y_2) or in (x_1, y_2) and (x_2, y_1)[2]. This problem is often referred to as *binding problem.* What we need in order to represent two points or even the whole triangle is a mechanism that binds the appropriate x- and y-coordinates together.

The problem becomes even more apparent in a second example. The word *tree* can be viewed as a structured object, namely a sequence, composed of the three elements *t, r,* and *e,* with *e* occurring twice in the structure. Again, the simultaneous representation of the three objects is insufficient as a representation of the whole word, because it provides no information about the order in which the letters occur in the sequence. Moreover, there is no information indicating that the element *e* is twice present in the structure. Consequently, we could not distinguish such a representation for *tree* from the representations for *etr*, *ter*, or *eter.*

What becomes clear from the two examples is that in order to represent structured objects a representational scheme needs to provide the means for two different operations:

- the binding of representations and
- the composition of objects from such bindings, individual elements, and smaller objects, i.e., the simultaneous representation of objects' constituents.

If we can represent the bindings (x_1, y_1), (x_2, y_2), and (x_3, y_3) of the triangle coordinates, then the simultaneous representation of these bindings unambiguously represents the whole triangle. The same works for the second example. If we can represent the binding of *t* to some element *first_letter*, of *r* to an element *second_letter*, and so on, then the simultaneous activation of these bindings represents the word *tree.* Binding the characters to elements representing the positions in a word also solves the problem of multiple occurrences. In the word *tree* the element *e* can simply be bound twice in the structure, once to *third_letter* and once to *fourth_letter.*

Both local and distributed representation support the simultaneous representation of objects, but the binding of objects is problematic. In a local representation each unit represents exactly one object. In order to represent more than one object at a

[2]This example is adapted from Hinton et al. (1986).

time we simply activate all units for the objects we want to represent. However, for the representation of bindings a new unit has to be introduced for each binding we wish to represent. With an increasing number of elements in a domain, the number of possible bindings and hence the number of units needed for their representation becomes prohibitive due to the exponential growth.

Simultaneously representing two or more objects in distributed fashion can be realised by superimposing the patterns of activity that represent the individual objects. Due to the amount of redundancy present in distributed representations, the objects will still be identifiable. However, a distributed representation does not provide a mechanism as such to represent the binding of two or more objects.

Role Decomposition

We will see in Sections 2.2 to 2.5 that most connectionist schemes for the representation of structured objects are based on the binding of fillers to roles as already suggested by Hinton (1981) for the representation of predicates. A prerequisite of such representational schemes is a suitable decomposition of the structures into roles and fillers. How structures are best decomposed depends on the task the structures are used for. A string of length n, for example, can in the simplest case be viewed as having n roles $\{r_1, r_2, \ldots, r_n\}$, where r_i is the role of the ith element in the string. As we have seen above, with this role decomposition the word *tree* is represented by binding the filler t to the role r_1, r is bound to r_2, and so on. However, roles encoding the positions of characters are not the only possible roles for the representation of a string.

A different, less obvious role decomposition formed, for example, the basis of the 'Wickelfeature' representation used by Rumelhart and McClelland (1986) in a connectionist system that learns to form the past tense of English verbs. In this mechanism roles do not encode the position of a character in a string but the context it appears in. Smolensky (1990) refers to these kind of roles as *contextual* roles. For the representation of the word *tree*, the element t is bound to a role *'preceded by nothing and followed by r'*, r is bound to *'preceded by t and followed by e'*, and so on. This role decomposition introduces more ambiguities for the representation of a single structure than the use of purely positional roles. For example, now the string *ababa* cannot be distinguished from the string *aba,* and a system employing this role decomposition

would not be able to correctly represent the word *banana*. The use of contextual roles can in some cases solve the binding problem, however. Whereas the simultaneous activation of the bindings (r_1, a), (r_2, b), (r_1, x), (r_2, y) potentially represents the strings *ab* and *xy* or the strings *ay* and *xb,* using contextual roles here disambiguates the representations. Moreover, roles that encode some of the context an object appears in can provide a connectionist network with information that captures the regularities of objects needed to solve context-sensitive tasks (Smolensky, 1990).

Note that the roles in a structure do not always have to be represented explicitly. In a symbolic system operating on strings, for example, strings are represented by ordered sets of characters. This order implicitly determines the role each character in the set is bound to; the first character is in first position, the second character in second position, and so on. Explicitly representing both fillers and roles, however, allows us to represent the string as an unordered set of bindings.

Concatenative and Functional Compositionality

We have established so far that the representation of structured objects requires mechanisms that are able to compose a structure from smaller objects by binding components like roles and fillers together and by simultaneously representing the structure constituents. This is true for both connectionist and symbolic representations. As outlined in Chapter 1, Connectionists and Symbolists do agree that representations underlying high-level cognitive processes need to be compositional; they disagree, however, over how to achieve this.

In systems based on symbolic representations the composition of structured objects is realised by the concatenation of symbols. Concatenation can be informally described as a method of ordering and linking successive constituents without altering them in any way as a composed expression is formed (van Gelder, 1990). What follows directly from this definition is that representations of structured objects based on concatenation exhibit an internal syntactic or constituent structure as defined by Fodor and McLaughlin (1990, p.186):

> "[F]or a pair of expression types E1 and E2, the first is a classical constituent of the second if the first is tokened whenever the second is tokened.

> For example, the English word 'John' is a classical constituent of the English sentence 'John loves the girl' and every tokening of the latter implies a tokening of the former (specifically, every token of the latter contains a token of the former; you can't say 'John loves the girl' without saying 'John')."

It is exactly this constituent structure of representations that according to Fodor and Pylyshyn (1988) forms the basis for systematic structure-sensitive processing of representations in symbol processing systems. They further argue that features of cognition such as productivity, systematicity, and compositionality are "explicable only on the assumption that mental representations have internal structure" (Fodor and Pylyshyn, 1988, p.33). But is constituent structure the only form of structure in representations that supports systematic processing? And is concatenation a suitable mode of composition for connectionist architectures?

One characteristic of representations for structures formed by concatenation is that they contain the full representations of all elements the structures are composed of. Consequently, the size of a representation increases when more elements are added to the structure it represents. This turns out to be problematic for connectionist architectures, because of the fixed number of resources available for the representation of objects. Connectionist architectures can therefore only employ concatenation for representations up to a predefined size.

However, van Gelder (1990) suggests an alternative to concatenation. He argues that connectionist architectures can facilitate *functional compositionality*:

> "Functional compositionality is obtained when there are general, effective, and reliable processes for (a) producing an expression given its constituents, and (b) decomposing the expression back into those constituents." (van Gelder, 1990, p. 361)

According to this definition the concatenation of symbols is one form of functional composition. However, nothing in the definition of functional compositionality suggests that the composition operation *must* preserve the full representations of all constituents in the representation of the composed object. What is crucial about such a mechanism is that there exists an inverse operation to the composition that can reliably

decompose a hierarchical object back into its components even if the representations of the components were changed during the encoding process. The representational schemes discussed in Sections 2.2 to 2.5 all implement modes of functional compositionality. The resulting representations do not possess a constituent structure in the sense that representations of constituents are not altered when combined with other constituents of a structure. We will see in the following Chapters, however, that they still support systematic structure-sensitive processing.

The notion of functional compositionality offers an alternative to the concatenation of representations that allows us to bind elements of an object together and to compose hierarchical structures from constituents. However, implementations of the functional composition in connectionist architectures still face the problem of restricted resources which is not automatically solved by discarding concatenation as the mode of composition. This can be seen on the example of the Gödel numbering scheme for formal languages which implements a merely functional form of composition (van Gelder, 1990). A Gödel number for an expression in a formal language is derived from the Gödel numbers of its components without simply concatenating the numbers representing the components together, and given the Gödel number of a complex expression we can decompose it into the Gödel numbers of its components by means of prime decomposition. However, although the Gödel number representation for a structure does not contain the full representations, i.e., the respective Gödel numbers, of the structure components, the size of a Gödel number still grows significantly with the size of the structure it represents. This again would make the numbering scheme as an implementation of functional compositionality unsuitable for connectionist architectures. The remaining question is therefore how to implement a compositional non-concatenative representational scheme in a connectionist architecture with fixed resources. This question can be answered by combining the concept of functional compositionality with the idea of reduced representation.

2.1.5 Reduced Representation

Hinton (1990) developed the concept of *Reduced Representation* as a general method of representing hierarchical structures using distributed representation. He observes that there are two ways of viewing representations of compositional objects within

a system. If a compositional object is the focus of attention in the system, then the object with all its components has to be fully represented. However, if the object is a component of another compositional structure which is the focus of attention at the time, then it can be represented in a reduced manner. In this case the components of the object need not be represented in full detail. The representation of the object only has to contain enough information about its components to allow for the recovery of their full representations when necessary. The representation of the object is called reduced.

To give an example, if we are given a piece of paper and asked to draw our home town viewed out of an airplane window on it, we will most likely draw the typical landmarks like a river or the surrounding hills and the principal layout of the town, the main square and some roads with houses on both sides. We might even sketch some cars on the roads and trees in a park. However, we will be unable to put every detail of every road into the picture; the colour of every front door, the makes of the parked cars, or the flowers in the front gardens. So if our aim is to represent the whole town on one piece of paper, we will have to represent at least some parts of it in an abstract, reduced manner. However, if we are then asked to draw the road we live in on the back of the same piece of paper, we will be able to expand the picture. We can now use the same space for a single road only, which will allow us to draw it in much more detail, including the colour of the neighbours' door, the cars typically parked in front of our house, the corner shop down the road, and so on. This side of the piece of paper will contain a full representation of the road.

Figure 2.2 illustrates the principle of reduced representation and the use of limited representational resources on a more formal example of a hierarchical structure. The tree on the left side of the picture is represented using the representational resources on the right side in different ways depending on the focus of attention at the time. The red lines show the use of the resources when **A** is the focus of attention. In this case it needs to be fully represented with all its components **B**, **C**, and **D**. However, the object **D** can be represented in a reduced manner, because we are not interested in the specifics of its components. The representation of **D** takes up only a part of all representational recourses (**A'** is the full representation of **A** and **D"** is the reduced representation of **D**.). The blue lines show the use of the resources when **D** is the focus

of attention. Now all resources are made available to represent **D** with its components **E**, **F**, and **G**. **D'** is the full representation of **D**. It now takes up the same resources as **A'** before.

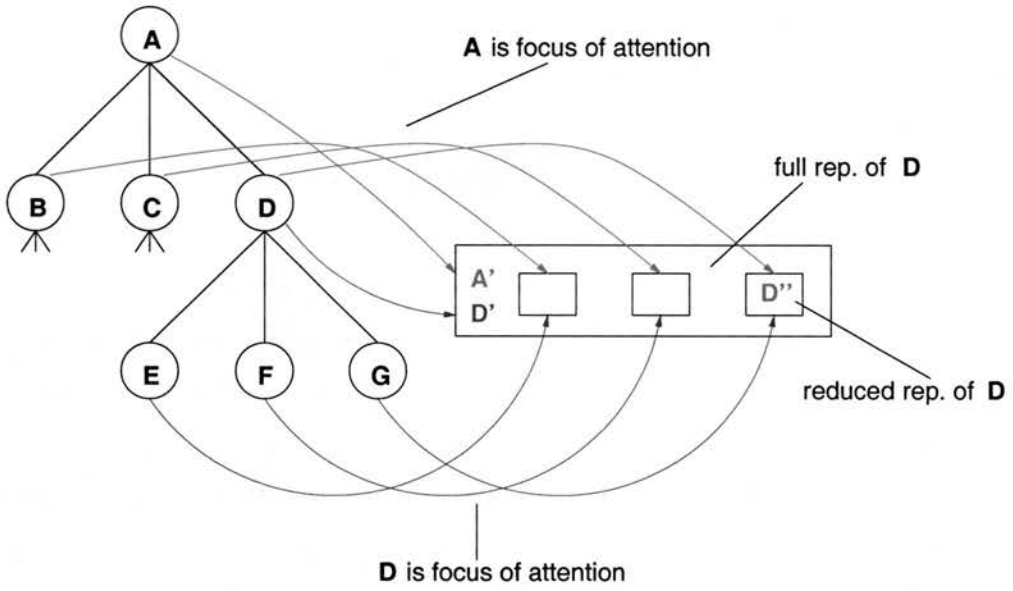

Figure 2.2: Representation of a hierarchical structure (left) using a fixed number of resources (right). The picture is adapted from Hinton (1990).

The picture is somewhat misleading in that it suggests that reduced representations take up a certain number of representational resources which are exclusive to them. This does not have to be the case. Reduced representations can be distributed over all resources available and superimposed. What is crucial, however, is that when an object is represented in a reduced manner it shares the representational resources with other objects represented in reduced representation. It will be allocated all representational resources available, however, if the system requires its full representation.

For reduced representations to be usefully implemented in a connectionist architecture they have to meet the following criteria (cf. Hinton, 1990; Plate, 1994):

- Reduction of size: A reduced representation of a composed object requires fewer representational resources than its full representation. This enables the composition of structured objects from the reduced representations of their components without an increase of representational resources. The representations for composed structures can be of the same size as the representations of their com-

ponents. Consequently, the same composition operations can be applied recursively to the components of an object and to the object itself in order to combine it with further objects to a larger hierarchical structure.

- Invertability of the encoding mechanism: For the operation that encodes objects in reduced representation there exists an inverse operation that can recover the full representation of the object from its reduced representation.

- Systematic relationship between reduced and full representations: The reduced representation of an object is systematically related to the full representation of the object such that the reduced representation provides some information about the components of the object without requiring the expansion to the full representation.

As we will see in the remainder of this Chapter, these criteria are met by representational schemes that implement the combination of distributed representations by means of functional compositionality. The redundancy present in distributed representations allows for the reduction of representations in size while still providing enough information to later recover their full representations. It further supports the error correction in noisy versions of representations expanded from their reduced forms. The ability to represent similar objects by similar distributed representations and the similarity preserving properties of the methods implementing the composition of hierarchical objects ensure the required systematic relationship between the resulting full and reduced representations.

It is important to note the difference between reduced representations and pointer based representations used in symbolic systems. The first two criteria above suggest that pointers could be viewed as reduced representations of the objects they point to. A pointer to an object is smaller than the full representation of the object and the reduction is invertible; once a pointer to an object has been created, we can follow it to the full representation of the object. However, pointers are usually chosen arbitrarily and do not carry any information about the object they point to other than its location. Hinton's reduced representation, however, requires a more systematic relation between full and reduced representations. As we will see in Chapter 3, the ability to gain information about a structured object from its reduced representation allows for very

efficient operations over structures that are not usually provided by symbolic systems. For example, reduced representations can be matched without expansion into full representations, providing information about the similarity of the internal structure of the objects they represent and the components these objects are composed of. In symbolic systems, on the other hand, structural matching of representations usually requires the decomposition and component-wise comparison of structures.

With distributed representation, functional compositionality, and reduced representation we have the three key concepts that enable us to represent hierarchical structured objects in connectionist architectures. Various implementations of these concepts have been proposed. In the remainder of this Chapter we will discuss four different schemes for the representation of hierarchical structures, Tensor Product Representation, Recursive Auto-Associative Memory, Holographic Reduced Representation, and Binary Spatter Code.

2.2 Tensor Product Representation

Tensor Product Representation was introduced by Smolensky (1990) and although it does not meet all criteria of reduced representations it is an interesting candidate for the representation of hierarchical objects in connectionist systems. In Tensor Product Representation two vectors representing a filler and a role within a structured object are bound by means of their tensor product. The tensor product of a n-dimensional vector $\mathbf{X} = (x_1, \ldots, x_n)$ and a m-dimensional vector $\mathbf{Y} = (y_1, \ldots, y_m)$ is the nm-dimensional vector $\mathbf{Z} = \mathbf{X} \otimes \mathbf{Y}$ that contains all products $x_i\, y_j$ for $i = 1 \ldots n$ and $j = 1 \ldots m$. The tensor product of two ordinary vectors is equivalent to the outer product of the two vectors and is called a tensor of rank 2. A simple vector can be viewed as tensor of rank 1. A tensor of rank 2 can again be combined with another tensor, leading to a tensor of a higher rank. This recursive use of a tensor allows for building recursive structured objects. Figure 2.3 shows the tensor product of two vectors with elements between 0 and 1. The shades of grey represent the values of the vector elements from 0 (white) to 1 (black). Such individual bindings of roles and fillers can be superimposed by vector addition to form hierarchical structures. Note, however, that a special tensor addition operation has to be defined for the superposition

of tensors with different rank.

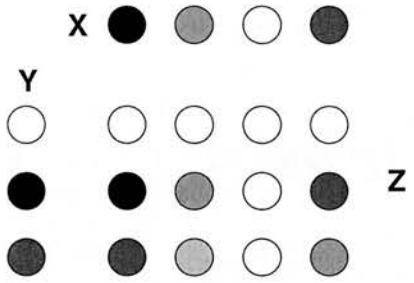

Figure 2.3: The tensor product $\mathbf{Z} = \mathbf{X} \otimes \mathbf{Y}$.

Under the condition that the representations for all roles in a structure are linearly independent, a set of *unbinding vectors* can be calculated from the role vectors in the domain such that the inner product of a tensor representing a role-filler binding and the unbinding vector corresponding to the role results in the filler. In this way, an element can be retrieved from a structure. The filler is recovered with complete accuracy, but this exact unbinding operation requires the calculation of the unbinding vectors first. Unbinding a filler is also possible using the role vector itself as the unbinding vector. However, the result of this operation is only completely accurate if all role vectors in the system are orthogonal to each other. If the role vectors are non-orthogonal, this unbinding method results in a noisy version of the original filler and a clean-up mechanism has to be provided to fully recover the filler from a binding. By using the approximate unbinding operation, a role can also be retrieved from the binding, in this case using the filler as the key.

Tensor Product Representation implements the concept of functional compositionality and meets some criteria of reduced representations. Hierarchical structures can be composed without the use of concatenation, and although the representation of a structure does not contain the full representations of all its elements, they can be retrieved using the unbinding operation. However, the size of a tensor increases with the size of the structure it is representing which prohibits the recursive application of the same composition operation to representations of objects with different complexity. Nevertheless, Tensor Products can still be used in connectionist architectures as long as the tensors are restricted to a certain rank as is shown, for example, by Dolan and Smolensky (1989) who used tensor products to re-implemented Touretzky and Hin-

ton's (1988) distributed connectionist production system which was originally based on a coarse coded distributed memory.

2.3 Recursive Auto-Associative Memory

Pollack (1988) developed a type of connectionist network which is able to learn representations of structured objects that do not increase with the size of the structures represented. He called such networks Recursive Auto-Associative Memories.

A RAAM is a three-layer feedforward network. The input and output layer of the network consist of k slots with n units each. The hidden layer has n units. Typical values for k are $k = 2$ (dual RAAM) and $k = 3$ (ternary RAAM). Input and hidden layer are fully connected as are hidden and output layer. The connections between the input layer and the hidden layer of the RAAM are often referred to as the encoder part of the network, the connections between the hidden layer and the output layer are the decoder part of the network.

Elements in a domain are represented by n-dimensional vectors and both local or distributed representations can be used. A RAAM is trained as an auto-associator using the backpropagation algorithm (Rumelhart et al., 1986; LeCun, 1985). k elements are presented to the input layer of the network. The network is trained to reproduce the representations of these elements in the output layer. This forces the network to develop a representation in the hidden layer that combines information about all k input elements. Since the hidden layer contains only as many units as one slot of the input layer, the new representation cannot contain the full representations of the input elements. Rather, the network develops a reduced description of the inputs in the hidden layer, which are expanded into the full representations again in the decoder part of the network. Since the newly acquired representation is of the same size as the representation of a single element, it can be further combined with other representations by feeding it back into one slot of the input layer of the network.

Figure 2.4 shows a dual RAAM with recursive connections from the hidden layer to one slot of the input layer which are activated during encoding and with recursive connections from one slot of the output layer to the hidden layer used in the decoding process. This network can learn representations of sequences or binary right-branching

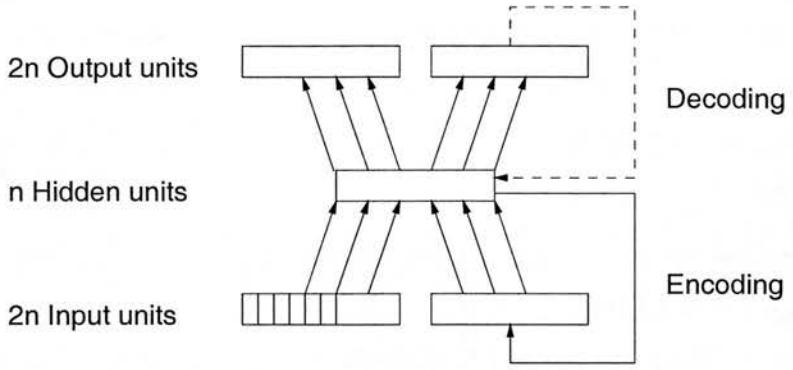

Figure 2.4: A dual RAAM for n-dimensional vectors with recursive connections for the encoding and decoding of hierarchical structures.

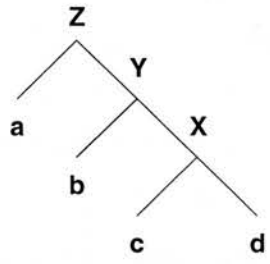

Figure 2.5: A recursive structure that can be encoded using the dual RAAM shown in Figure 2.4.

trees as exemplified in Figure 2.5. One training cycle for learning the encoding of this tree involves the following steps.

Time step	Input layer	Hidden layer	Output layer
1	$(\mathbf{c}, \mathbf{d})$	$\mathbf{X}$	$(\mathbf{c}', \mathbf{d}')$
2	$(\mathbf{b}, \mathbf{X})$	$\mathbf{Y}$	$(\mathbf{b}', \mathbf{X}')$
3	$(\mathbf{a}, \mathbf{Y})$	$\mathbf{Z}$	$(\mathbf{a}', \mathbf{Y}')$

First, vectors $\mathbf{c}$ and $\mathbf{d}$ are presented to the input layer of the network and as target for the output layer. The network forms the representation $\mathbf{X}$ in the hidden layer that combines information about both input elements. Using the recurrent connections, $\mathbf{X}$ is then presented as input together with the vector $\mathbf{b}$. The vector $\mathbf{Y}$ formed in the hidden layer now represents the combination of $(\mathbf{c}, \mathbf{d})$ and the element $\mathbf{b}$. Finally, $\mathbf{Y}$ and $\mathbf{a}$ are presented as input and target vectors and $\mathbf{Z}$ is formed in the hidden

layer representing the full tree. Note that the hidden layer representations change in each such training cycle. Since these representations form also part of the input and target patterns, training a RAAM can be viewed as a moving target learning problem (Pollack, 1990).

To decode the representation of a tree into its components, it is directly fed into the hidden layer of the RAAM. Since the network is trained as an auto-associator the output layer provides output vectors which resemble the k components of the tree. In our example, feeding $\mathbf{Z}$ into the hidden layer of the RAAM results in the vectors $\mathbf{a}'$ and $\mathbf{Y}'$ in the output layer. Since $\mathbf{Y}'$ still represents a composed object rather than an element of the domain, it is again fed into the hidden layer for further decomposition leading to $\mathbf{b}'$ and $\mathbf{X}'$ in the output layer. The process is once more repeated with $\mathbf{X}'$ resulting in the decoding of $\mathbf{c}'$ and $\mathbf{d}'$. Assuming successful training, the results in the output layer of the network are close enough to the original element vectors to be recognisable, i.e., $\mathbf{a}' \approx \mathbf{a}$, $\mathbf{b}' \approx \mathbf{b}$, etc.

A RAAM is able to learn the encoding and decoding of a number of such trees in parallel. Since the depth of the trees can vary, one needs a procedure to decide whether a decoded vector represents a single element or a composed object. Various procedures are in use depending on the exact implementation of the network and the input representation used. A commonly used method is to represent elements by binary vectors, whereas the representations formed in the hidden layer of the RAAM are real-valued vectors with elements that tend to be between but not close to 0 and 1 (Plate, 1997a). This difference can be used to distinguish elements from trees. A different method is adopted by Niklasson (1999). An extra bit functioning as an element/tree separator is added to the representation. It is set to 1 if a single element is represented and to 0 if the representation stands for a tree. This information is then used in the decoding process. The representation needs further decoding if the value of the extra bit is close to 0. Otherwise the representation stands for an element. This method does not only simplify the decision about the further decoding of representations significantly, Niklasson (1999) also reports that the representations formed in the hidden layer become more distinct, i.e., the RAAM makes better use of the representational space, which generally increases the encoding/decoding performance of the RAAM. Niklasson further maintains that a clearer separation of representations in the representational

space increases the suitability of the representations for systematic processing.

Once a vector is determined as a representation for an element, one still has to decide which element it actually refers to. A commonly used method is to calculate the Euclidean distance between the output vector and all target representations and to choose as result of the decoding the element with the closest distance to the output vector.

If an additional memory for structured objects is provided, the RAAM is not restricted to trees branching only to one side because representations formed in the hidden layer can be more flexibly fed back into the input layer. Further, input slots can be left empty by proving a special representation *nil*. With these extensions a ternary RAAM can, for example, learn the representation for a structured expression in propositional logic such as $x \rightarrow (\neg a \vee b)$ shown in Figure 2.6.

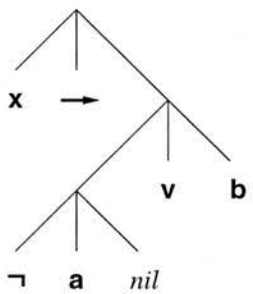

Figure 2.6: A recursive structure that can be encoded using a ternary RAAM with additional memory.

Like tensor products, representations formed in the hidden layer of RAAMs can be viewed as role-filler bindings, although here the roles are not represented explicitly. Rather, the connections between the input and hidden layer of the RAAM serve as roles and the vectors of the input activations represent the fillers. The matrix-vector multiplications of the weight matrix for the connections and the input vectors perform the role-filler bindings (Plate, 1997b).

2.4 Holographic Reduced Representation

Holographic Reduced Representation (Plate, 1994) is a representational scheme that uses circular convolution and vector addition to combine vectors representing elements

of a domain in hierarchical structures. Elements are usually represented by randomly chosen high-dimensional vectors. A vector representing a structure is of the same size as the vectors representing the elements it contains. Thus, it can be recursively used as part of a larger structure without increasing the size of the representational space.

Circular convolution binds two n-dimensional vectors $\mathbf{X} = (x_0, \ldots, x_{n-1})$ and $\mathbf{Y} = (y_0, \ldots, y_{n-1})$ to a n-dimensional vector $\mathbf{Z} = (z_0, \ldots, z_{n-1})$ such that

$$\mathbf{Z} = \mathbf{X} \circledast \mathbf{Y} \quad \text{with} \tag{2.6}$$

$$z_j = \sum_{k=0}^{n-1} x_k \, y_{j-k} \quad \text{for} \quad j = 0, \ldots, n-1 \tag{2.7}$$

where subscripts are modulo-n. Circular convolution is commutative, associative, and it distributes over vector addition. Let $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$ be HRR vectors of the same dimensionality. Then

$$\mathbf{X} \circledast \mathbf{Y} = \mathbf{Y} \circledast \mathbf{X} \tag{2.8}$$

$$(\mathbf{X} \circledast \mathbf{Y}) \circledast \mathbf{Z} = \mathbf{X} \circledast (\mathbf{Y} \circledast \mathbf{Z}) \tag{2.9}$$

$$\mathbf{X} \circledast (\mathbf{Y} + \mathbf{Z}) = \mathbf{X} \circledast \mathbf{Y} + \mathbf{X} \circledast \mathbf{Z} \, . \tag{2.10}$$

There is an identity vector $\mathbf{I} = (1, 0, 0, \ldots)$ and a zero vector $\mathbf{0} = (0, 0, 0, \ldots)$ with $\mathbf{X} \circledast \mathbf{I} = \mathbf{X}$ and $\mathbf{X} \circledast \mathbf{0} = \mathbf{0}$.

Figure 2.7 illustrates the circular convolution of two vectors with three elements. A number of such vector bindings are combined to a single structure by the superposition of the vectors representing the individual bindings. Vector addition is used as superposition operation.

Under the condition that the elements of each n-dimensional vector are independently and identically distributed as $\mathrm{N}(0, 1/n)$, circular correlation is an approximate inverse to circular convolution and can be used for retrieving elements from bindings. For the binding in Equation (2.6) we get

$$\tilde{\mathbf{X}} = \mathbf{Z} \oplus \mathbf{Y} \quad \text{with} \tag{2.11}$$

$$\tilde{x}_j = \sum_{k=0}^{n-1} z_k \, y_{j+k} \quad \text{for} \quad j = 0, \ldots, n-1 \tag{2.12}$$

where subscripts are modulo-n. However, decoding a binding by circular correlation produces only approximate versions of the original vectors, i.e., $\tilde{\mathbf{X}} \approx \mathbf{X}$. Thus, a

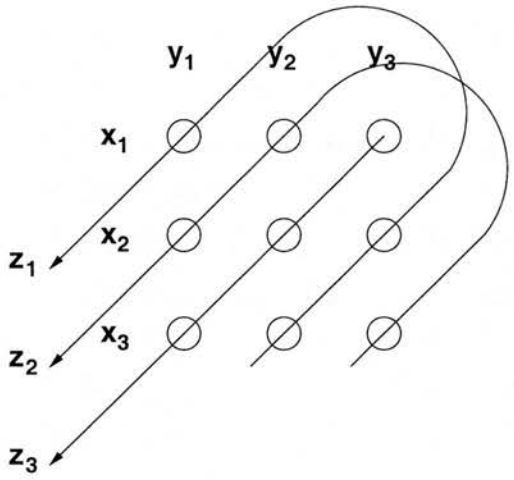

Figure 2.7: Circular convolution of two vectors with three elements. The picture is adapted from Plate (1994).

clean-up memory containing all possible elements of a domain is needed in order to fully restore element vectors from bindings.

Since circular correlation distributes over vector addition, elements can be retrieved from complex structures directly. For example, we can recover the vector $\mathbf{X}$ from the structure

$$\mathbf{Z} = \mathbf{X} \circledast \mathbf{Y} + \mathbf{A} \circledast \mathbf{B} \tag{2.13}$$

by correlating $\mathbf{Z}$ with the vector $\mathbf{Y}$:

$$(\mathbf{X} \circledast \mathbf{Y} + \mathbf{A} \circledast \mathbf{B}) \oplus \mathbf{Y} = \tag{2.14}$$

$$\mathbf{X} \circledast \mathbf{Y} \oplus \mathbf{Y} + \mathbf{A} \circledast \mathbf{B} \oplus \mathbf{Y} = \tilde{\mathbf{X}} + \underbrace{\mathbf{A} \circledast \mathbf{B} \oplus \mathbf{Y}}_{noise}. \tag{2.15}$$

As seen above, $\tilde{\mathbf{X}}$ is a noisy version of $\mathbf{X}$. Since $\mathbf{A} \circledast \mathbf{B} \oplus \mathbf{Y}$ is not likely to be similar to any item in the clean-up memory, it can be treated as extra noise. Thus, $\tilde{\mathbf{X}} + \mathbf{A} \circledast \mathbf{B} \oplus \mathbf{Y}$ will be decoded as $\mathbf{X}$ in the clean-up memory.

The circular correlation $\mathbf{X} \oplus \mathbf{Y}$ is equivalent to the circular convolution $\mathbf{X}' \circledast \mathbf{Y}$. $\mathbf{X}'$ is called *involution* of $\mathbf{X}$ and is defined as $x'_i = x_{-i}$ for $i = 0 \ldots n-1$, where subscripts are modulo-n for n-dimensional vectors. Since circular correlation is neither commutative nor associative, it is often preferable to use the involution and circular convolution instead of correlation for the decoding of structures (Plate, 1994). This

way encoding and decoding of complex HRR structures can be implemented using vector addition and circular convolution only. This is shown in the following example. Let { **X, Y, op, not, disj, first, second, impl, ante, cons** } be a set of HRR vectors of the same dimensionality representing the elements X, Y, *operation, negation, disjunction, first operand, second operand, implication, antecedent*, and *consequent*[3]. Then structures representing the logically equivalent formulae $X \rightarrow Y$ and $\neg X \vee Y$ can be constructed as follows:

$$\begin{aligned}
\textbf{not_X} &= (\,\textbf{not} \circledast \textbf{X}\,) \\
\textbf{X_impl_Y} &= (\,\textbf{op} \circledast \textbf{impl} + \textbf{ante} \circledast \textbf{X} + \textbf{cons} \circledast \textbf{Y}\,) \\
\textbf{not_X_or_Y} &= (\,\textbf{op} \circledast \textbf{disj} + \textbf{first} \circledast \textbf{not_X} + \textbf{second} \circledast \textbf{Y}\,)\,.
\end{aligned}$$

One can interpret circular convolution as performing a role-filler binding, for example, of the fillers X and Y to the roles *antecedent, consequent, first operand,* and *second operand.* In order to retrieve, e.g., the filler X from the disjunction we correlate the structure with both the vectors **not** and **first**. Alternatively, we can convolve it with the approximate inverses **not**$'$ and **first**$'$ of the roles.

The circular convolution of two vectors $\mathbf{X}$ and $\mathbf{Y}$ can also be expressed as the multiplication of the matrix $\mathrm{circ}(\mathbf{X})$ and the vector $\mathbf{Y}$. $\mathrm{circ}(\mathbf{X})$ is a *circulant* matrix (Davis, 1979) derived from the vector $\mathbf{X} = (x_0, \ldots, x_{n-1})$:

$$\mathrm{circ}(\mathbf{X}) = \mathrm{circ}(x_0, x_2, \ldots, x_{n-1}) \tag{2.16}$$

$$= \begin{pmatrix} x_0 & x_1 & \ldots & x_{n-1} \\ x_{n-1} & x_0 & \ldots & x_{n-2} \\ \vdots & \vdots & \ldots & \vdots \\ x_1 & x_2 & \ldots & x_0 \end{pmatrix}. \tag{2.17}$$

The elements of each row of $\mathrm{circ}(\mathbf{X})$ are identical to those of the previous row, but are shifted one position to the right and 'wrapped around'. With $\mathrm{circ}(\mathbf{X})$ we get

$$\mathbf{Z} = \mathbf{X} \circledast \mathbf{Y} \tag{2.18}$$

$$\mathbf{Z} = \mathrm{circ}(\mathbf{X})\,\mathbf{Y}\,. \tag{2.19}$$

[3]Throughout this thesis we will print concepts in *italic* and vectors representing concepts in **bold**.

If $\text{circ}(\mathbf{X})$ is a non-singular matrix, then there exists an exact inverse operation to circular convolution, because

$$\mathbf{Y} = (\text{circ}(\mathbf{X}))^{-1}\,\mathbf{Z}. \tag{2.20}$$

The first column of $(\text{circ}(\mathbf{X}))^{-1}$ is the exact inverse $\mathbf{X}^{-1}$ of $\mathbf{X}$, i.e., $\mathbf{X}^{-1} \circledast \mathbf{X} = \mathbf{I}$. Thus, if $\mathbf{Z} = \mathbf{X} \circledast \mathbf{Y}$ and $\text{circ}(\mathbf{X})$ is non-singular, then $\mathbf{Y}$ can be exactly decoded by $\mathbf{Z} \circledast \mathbf{X}^{-1}$. However, Plate observes that "using the exact inverse for decoding results in a lower signal-to-noise ratio in the retrieved vector when the memory trace is noisy or when there are other vectors added to it" (Plate, 1994, p.82). Circular correlation is therefore preferred as the decoding operation. To further minimise the noise in the decoding process, HRR requires high-dimensional vectors even for small problems. In most experiments reported in the following Chapters we used vectors with 4096 elements. However, circular convolution can be efficiently computed using Fast Fourier Transforms. The Fourier Matrix $\mathbf{F}$ diagonalises a circulant matrix $\text{circ}(\mathbf{X})$ (Davis, 1979) such that

$$\text{circ}(\mathbf{X}) = \mathbf{F}^{-1}\text{diag}(\mathbf{F}\mathbf{X})\mathbf{F}. \tag{2.21}$$

Therefore,

$$\begin{aligned} \mathbf{X} \circledast \mathbf{Y} &= \text{circ}(\mathbf{X})\,\mathbf{Y} && (2.22)\\ &= \mathbf{F}^{-1}\text{diag}(\mathbf{F}\mathbf{X})\mathbf{F}\mathbf{Y} && (2.23)\\ &= \mathbf{F}^{-1}((\mathbf{F}\mathbf{X}) .* (\mathbf{F}\mathbf{Y})) && (2.24) \end{aligned}$$

where $.*$ denotes the element-wise multiplication of two vectors. Equation (2.24) can be calculated by $O(n \log n)$ operations for n-dimensional vectors.

Binding by circular convolution is what Plate (1994) calls *randomising*. The binding of a role to a filler is usually not similar to either the role or the filler. However, circular convolution preserves similarity between bindings with similar components. $\mathbf{A} \circledast \mathbf{B}$ is similar to $\mathbf{C} \circledast \mathbf{D}$ to the extent that $\mathbf{A}$ is similar to $\mathbf{C}$ and $\mathbf{B}$ is similar to $\mathbf{D}$. Plate (1994) refers to this kind of similarity as *structured similarity*. Superposition, on the other hand, preserves *unstructured similarity*. The superposition of two vectors is similar to both of them. With these complementary properties of circular convolution

and superposition, HRR for more complex structured objects are similar to the degree that they are structurally similar and contain the same or similar roles and fillers.

Plate's (1994) originally proposed mechanism requires the normalisation of HRR vectors, i.e., the length of all vectors, element vectors as well as vectors representing structures, should be 1. There are two reasons for normalisation: the dot product of two vectors reflects their similarity rather than their magnitudes and superimposed vectors are represented with equal weight in a structure. However, for some operations on HRR vectors, such as learning a vector that transforms a number of HRR structures at once, better results are achieved without normalisation. The reason is that such operations involve exact inverses of HRR vectors as will be shown in detail in Chapter 4. The expected length of an exact inverse to a HRR vector is often larger than the length of the vector itself, however (Plate, 1994). Note that when no normalisation is performed, we have to use the normalised dot product in order to measure the similarity of transformation vectors, and transformed structures should be normalised before decoding to ensure the correct retrieval of their elements.

Since a HRR vector representing a structure has the same mathematical properties as a vector representing an element, structures of unlimited size can in principle be processed. Such structures can be practically implemented by means of chunking (Plate, 1994). Chunking requires the storage of not only element representations but representations of more complex constituents in the clean-up memory. This allows for cleaning up intermediate results when decoding very large structures.

HRR shows excellent scaling properties (Plate, 1994):

- The number of individual bindings that can be stored in a structure representation of the form $(\mathbf{a} \circledast \mathbf{b} + \mathbf{c} \circledast \mathbf{d} + \ldots)$ grows approximately linearly with the vector dimension n.
- The probability of decoding errors for elements from such structures is negative exponential in n.
- The number of individual items that can be stored in the clean-up memory is exponential in n.

Plate (1994, Appendix D) gives a lower bound on the capacity of a convolution memory that stores k unordered bindings of distinct vectors drawn from a pool of m

element vectors of dimension n. This lower bound can be approximated for k in the range $(2 \ldots 14)$ and m in $(10^2 \ldots 10^{10})$ by

$$n = 4.5(k + 0.7) \ln \frac{m}{30q^4} \tag{2.25}$$

where q in the range of $(10^{-2} \ldots 10^{-10})$ is the probability of errors in the decoding process. The number of elements that can be reliably decoded from a structure representation decreases, however, when similar element vectors are used and when they are embedded in higher-order bindings. The latter means that more elements can be stored in a structure like $(\mathbf{a} \circledast \mathbf{b} + \mathbf{c} \circledast \mathbf{d} + \ldots)$ than in a structure of the form $((((\mathbf{a} \circledast \mathbf{b}) \circledast \mathbf{c}) \circledast \mathbf{d}) \ldots)$ using HRR vectors of the same size. The reason for this difference is that the noise when decoding elements from a higher-order convolution is larger than the noise when decoding elements from a lower-order convolution (Plate, 1994). Consequently, representing shallow hierarchies requires fewer representational resources than the representation of deep structures. We will see in Chapter 5 that representations of shallow hierarchies further allow for more flexible holistic operations than representations of deeper structures.

Note finally that HRR is closely related to other forms of convolution-based associative memories such as the non-linear correlograph (Willshaw et al., 1969) and TODAM (Murdock, 1982, 1993) and to matrix-based associative memories like the associative net (Willshaw et al., 1969) and Tensor Product Representation. Willshaw (1971) observes that all non-local associative memories can be described as

$$\mathbf{A} = \mathbf{M}^{-1}\mathbf{ZMB} \tag{2.26}$$

or in the non-linear case as

$$\mathbf{A} = [[\mathbf{M}^{-1}\mathbf{ZM}]\mathbf{B}]\,. \tag{2.27}$$

where $\mathbf{Z}$ is the memory matrix, $\mathbf{M}$ is a non-local transformation matrix, and $\mathbf{B}$ is the key to retrieve $\mathbf{A}$ from the memory matrix. The parenthesis [] represent non-linear operations. A particular instance of an associative memory is then defined by the choice of $\mathbf{M}$ and $\mathbf{Z}$. For HRR, the transformation matrix is the Fourier Matrix and the memory matrix $\mathbf{Z}$ is a diagonal matrix, leading to the representation of bindings without increase in the representational space. Matrix-based memories like the associative net and Tensor Product Representation on the other hand will have a different transformation matrix and a memory matrix $\mathbf{Z}$ with n^2 non-zero entries.

2.5 Binary Spatter Code

Binary Spatter Code (Kanerva, 1997) is a connectionist representational scheme for hierarchical structured objects that is very similar to Holographic Reduced Representation. It is often even referred to as a form of HRR. However, vectors and operations differ from the ones introduced in the previous Section. Objects are represented by high-dimensional binary vectors. The vector elements are identically and independently distributed, so 0 and 1 are equally probable values of a vector element. Two n-dimensional vectors $\mathbf{X} = (x_1, \ldots, x_n)$ and $\mathbf{Y} = (y_1, \ldots, y_n)$ with $x_i, y_i \in \{0, 1\}$ are combined to a n-dimensional vector $\mathbf{Z} = (z_1, \ldots, z_n)$ with $z_i \in \{0, 1\}$ by bit-wise Boolean Exclusive OR (XOR). Like circular convolution in HRR, bit-wise XOR is randomising, that is, $\mathbf{Z}$ is not similar to either $\mathbf{X}$ or $\mathbf{Y}$. Further, XOR is its own inverse operation. Thus, in order to retrieve the element vector $\mathbf{X}$ from the structure $\mathbf{Z}$, $\mathbf{Z}$ is combined by bit-wise XOR with the element vector $\mathbf{Y}$. The superposition of vectors is realised by the normalised sum of the vectors which is calculated by the bit-wise majority rule with ties broken at random. Like the vector addition in HRR, the superposition of BSC vectors preserves unstructured similarity. The result of the superposition is similar to both input vectors. Decoding an element directly from a composed structure results in noisy versions of the original vectors. In this case a clean-up mechanism is needed.

Despite its mathematical simplicity, BSC is a powerful mechanism for representing hierarchical structures in connectionist architectures. In fact, the mathematical simplicity of the scheme is a very desirable property, because it allows for a straightforward implementation and detailed analysis of representations and operations involved. BSC does have some drawbacks in comparison to HRR, however. Most notably, using the bit-wise majority rule for superposition discards a lot of information in the vectors (Kanerva, 1998). As an alternative Kanerva (1998) uses the actual sum of the vectors and replaces 1 and 0 in the binary vectors with 1 and -1, respectively. A number of m bindings of n-dimensional BSC vectors $\mathbf{X}$ and $\mathbf{Y}$ is then stored in a n-dimensional vector $\mathbf{Z}$ with

$$z_i \;=\; -\sum_{j=1}^{m} x_i(j)\, y_i(j)\,. \tag{2.28}$$

A single vector $\mathbf{X}(k)$ is retrieved from this binding by

$$\tilde{x}_i(k) = -z_i\, y_i(k) \tag{2.29}$$

$$= \sum_{j=1}^{m} \Big(x_i(j)\, y_i(j) \Big)\, y_i(k) \tag{2.30}$$

using $\mathbf{Y}(k)$ as the key. This revised representational scheme was used, for instance, for the transformation of a number of different structures (Kanerva, 1998) as will be discussed in detail in the following Chapter.

An interesting question regarding the relation between BSC and HRR is that of storage capacity. Given the mathematical simplicity of BSC vectors, they can be easily increased in size which is generally advantageous in terms of storage capacity and retrieval accuracy. So would it be more efficient to use longer binary BSC vectors or shorter real-valued HRR vectors? The field still lacks a formal quantitative comparison of HRR and BSC and such comparison might be problematic as the vectors differ in two parameters: their length and the nature of their elements (real/binary). This makes it difficult to isolate the causes of any differences between the two schemes. Moreover, this comparison would not hold for the more often used modified BSC vectors and operations discussed above. However, one way of relating the modified BSC to HRR is to compare their signal to noise ratios which can be taken as a measure of fidelity of recall. For m individual bindings of n-dimensional BSC vectors as given in Equation (2.28), the signal to noise ratio is $1/(m-1)$. This is the same signal-to-noise ratio as observed for HRR (Willshaw, 1971).

2.6 Summary

A prerequisite for the successful application of connectionist networks to cognitive processing is an adequate connectionist representation of all information relevant to the tasks at hand. Both local and distributed representations have their place in connectionist architectures. Local representation often requires large amounts of representational resources but is easily interpretable and analysed and therefore particularly suitable for the representation of input and output information in a connectionist network. Distributed representation is a more efficient way of representing information and is able

to capture the similarity of represented objects. This enables neural networks to generalise knowledge acquired during learning to previously unseen examples.

Any system, whether symbolic or connectionist, that is aimed at modelling high-level cognition must be able to represent hierarchical structured objects. We have seen that the simultaneous representation of the constituents of a hierarchical object is not sufficient to capture its internal structure. Rather, a representational scheme for hierarchical structures must facilitate special mechanisms that first bind components of structures together and then simultaneously represent these bindings.

Four schemes for the distributed representation of hierarchical structures have been discussed in this Chapter: Tensor Products, RAAM, HRR, and BSC. These schemes implement the concept of reduced representation and facilitate a functional rather than concatenative compositionality as found in symbolic systems. Despite some differences in their implementations, these schemes have much in common and can be viewed within a common framework for distributed representations of compositional structures (Plate, 1997a). All schemes are based on role-filler bindings and employ a binding operation (tensor product, circular convolution, matrix-vector multiplication, bit-wise XOR) and a superposition operation (vector addition, normalised sum of binary vectors) as two means of composing structured objects from elements in a domain. These operations preserve similarity such that the resulting representations for objects are similar to the extend that the objects are structurally similar and are composed of the same or similar elements. We will see in the following Chapters that this shared property of all schemes enables the structure-sensitive yet holistic processing of such representations for hierarchical structures.

Chapter 3

Holistic Operations on Connectionist Representations of Structures

In the previous Chapter we introduced a number of representational schemes that provide the means for the representation of hierarchical structures in a way suitable for connectionist processing. These representational mechanisms contradict one of the Symbolists' criticisms of connectionist architectures, namely that the lack of concatenative compositionality prevents such architectures from adequately representing hierarchical structures as required in high-level cognitive processing. However, as argued earlier, representations of structured objects are only truly useful if they also allow for systematic structure-sensitive processing. Such processes are straightforwardly implemented in symbolic systems making use of the fact that the representations of structures are themselves syntactically structured and contain full representations of all elements the represented objects are composed of. The head of a list, for example, will be found as the first constituent in the symbolic representation of the list.

We have seen in the previous Chapter that connectionist representations of structures are of profoundly different nature, however. With the use of reduced representations and the superposition of constituent representations they do not exhibit a clear syntactic structure. Looking at the representation of a list distributed over a number of connectionist units, we will find that all units contribute to the representation of the head, not only the first few ones. Consequently, these representations call for forms of structure-sensitive processing that do not rely on the syntactic structure of the repre-

sentation themselves.

It has been argued that the connectionist representational schemes introduced in the previous Chapter can support structure-sensitive processing of the representations they provide (e.g. Niklasson and Bodén (1997); Plate (1994, 1997a)). The binding and superposition operations used in the composition processes of these schemes result in representations of structures which, although not syntactically structured themselves, are systematically related to the representations of the elements they are composed of. This means that we can gain information about elements and their structural relation directly from the representations of the hierarchical structures without decomposing them. In other words, once representations for structures have been formed, one can operate on them *holistically.* They can be structurally mapped and even transformed by processes which do not require any decomposition of structures into their components.

In this Chapter we discuss a number of experiments presented in the literature and some of our own comparable simulations that investigate holistic operations on a variety of connectionist representations of hierarchical structures. We discuss the holistic classification, matching, and transformation of representations formed in the hidden layers of RAAMs and the holistic matching and transformation of HRRs, BSCs, and Tensor Products. We will start out with describing the tasks performed and the architectures and representations used and will then compare different results, examine the suitability of the representational schemes for particular holistic operations, and explore their strengths and weaknesses with respect to holistic processing in general.

3.1 Two Definitions

Although holistic operations have been investigated for quite some time there still seems to be a lack of a clear definition of holistic processing. For the purpose of this thesis, we will define holistic operations as follows:

Definition: Holistic Operation

A *holistic operation* is an operation applied to representations of structured objects that can act in one step on all constituents of an object without the need for its explicit

decomposition.

This definition is close to the one provided by Hammerton (1998) which stresses that in a holistic operation no search to locate or access constituents is required in order to operate on them. We also agree with Hammerton that holistic computations are not exclusive to connectionist architectures. Imagine an array of fixed length containing a number of symbols representing *As* which need to be changed into *Bs*. One could perform this symbolic operation holistically in a parallel symbolic architecture, given the size of the array is known in advance, i.e., no counting or search for constituents of the array is required. What makes connectionist networks particularly suitable for holistic operations, however, is that they can "act holistically on their input vectors by the nature of their design" (Hammerton, 1998). This allows for the application of the same holistic process to representations of hierarchical objects which are different in size and complexity, as long as these objects can be represented as input vectors to the same connectionist network.

Most of the holistic operations discussed in the following Sections are performed on a set of representations for hierarchical objects that are structurally equivalent, e.g., logical expressions such as $x_1 \rightarrow x_2$ with $x_1, x_2 \in \{a, b, c\}$. We will refer to such objects as classes of structures.

Definition: Class of Structures

The set of n hierarchical structured objects $\{A_i, A_2, \ldots A_n\}$ is called a *Class of Structures*, if any two objects $\{A_i, A_j\}$ with $i, j = 1 \ldots n$ and $i \neq j$ are structurally equivalent, i.e., they share all roles and the relations among them, but differ in at least one filler.

3.2 Holistic Operations on RAAM Structures

3.2.1 Classification of Structures

Blank et al. (1992) investigated different holistic operations on structures formed in a *sequential* RAAM. A sequential RAAM is a dual RAAM that encodes all hierarchical

structures as ordered lists or sequences of elements. For example, the sentence "John loves Helen" is encoded as the list

[John [loves [Helen *nil*]]].

The restriction of all possible structures to lists has the advantage that one of the input slots of the RAAM is exclusively used to represent elements whereas the second slot always represents an already composed structure. Consequently, the input slot for the composed structure and hence the hidden layer of the RAAM can be larger than the input slot for the single elements. Increasing the number of processing units in the hidden layer of the RAAM in turn increases the storage capacity of the network.

Blank et al. (1992) trained a sequential RAAM with 27 input units for elements and 30 input and hidden units for structures to encode simple English-like sentences of the forms noun-verb and noun-verb-noun. 341 sentences were generated from 15 nouns and 11 verbs and words were encoded using a localist binary representation. The nouns and verbs were described by lexical categories such as 'noun-animate' (for words like *boy, Jane, chimp*), 'noun-hunter' (*Tarzan, Jane*), 'noun-edible' (*banana, berry*), or 'verb-perceive' (*see, smell*), 'verb-hunt' (*hunt*), and 'verb-agress' (*kill, chase*). After successfully encoding and decoding all sentences in the RAAM, a three-layer feed-forward network was trained to classify 50 sentences depending upon the presence or absence of a certain feature in the sentence. For example, the classification network had to detect all sentences including a word of the category 'noun-aggressive'. After training, the network was able to generalise the classification to 50 novel sentences with 88% accuracy. When the classification network was taught to detect sentences containing two words of the categories 'noun-aggressive' and 'human', it generalised with 84% accuracy. However, for more complex classification tasks such as the detection of the same subject and object in one sentence, this simple architecture did not show any generalisation capacity.

Niklasson and Bodén (1997) report a very high generalisation performance for a simple perceptron trained on the holistic classification of binary trees encoded by a RAAM. 325 tree structures of the forms illustrated in Figure 3.1 were encoded using a 20-10-20[1] dual RAAM. The leaves could take on five different values which were

[1]The notation 'x-y-x RAAM' refers to a RAAM with x input and output units and y hidden units.

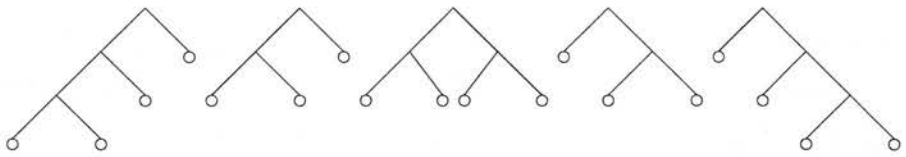

Figure 3.1: Five different types of trees learned by a dual RAAM.

encoded in a localist binary representation. In several tests the RAAM was trained to encode and decode these trees until a stable summed square error was reached after which the network decoded between 77% and 83% of the elements correctly. Two simple perceptrons, a balance classifier and a depth classifier, were then trained to classify the trees as being left-, right- or well-balanced and as having depth 2 or 3. After both classifiers were trained on half the tree structures they correctly classified between 95% and 100% of all trees.

Sperduti et al. (1995) combined a *labelling* RAAM (LRAAM) with a perceptron to learn the classification of logical terms that can be represented by labelled directed acyclic graphs. A LRAAM is an extension of the RAAM architecture. Extra units are added to the input and output layer of the RAAM that allow the representation of a label to be associated with the elements represented in the usual slots for inputs and outputs. Figure 3.2 shows a dual LRAAM with $2n + l$ input and output units, the l units carrying the label. Like a standard RAAM, the LRAAM is trained as an auto-associator and the representations formed in the hidden layer can be recursively used as part of the input.

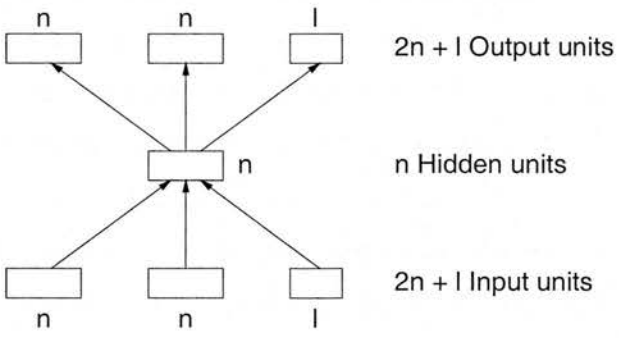

Figure 3.2: A dual labelling RAAM for encoding binary trees with labels. The l input and output units carry the representation for the label associated with the elements represented in the two input and output slots at the same time.

Using a labelled version of an RAAM extends the range of graphs that can be learned from relatively simple tree structures to cyclic graphs with labels attached to each node. Sperduti (1994) points out that LRAAMs also provide better solutions to the problems of terminating the learning and the decoding process. With the use of fixed representations for labels it should further be possible to dispense with the fixed pre-defined representations for elements necessary for successful training in conventional RAAMs. While fixed element representations in RAAMs are required to prevent collapsing all vectors to 0 during training (Paccanaro and Hinton, 2001), this role can be played by fixed labels in an LRAAM, thus potentially allowing for element representations to be learned. However, such learning was not implemented in the experiment described below.

Sperduti et al. (1995) trained a dual LRAAM to encode logical terms of different complexity. Typical terms were, for example, $f((i(a), i(b))$, $f(g(a, b), b)$, and $f(c, f(a, b))$. Different classification tasks were then defined over such terms ranging from detecting a particular element within a term to detecting instances of $f(X, X)$. Note that the latter is a much more difficult classification task than the tasks considered in the previous experiments. Here the classifier has to be able to compare arbitrary sub-terms corresponding to X, a task similar to the detection of the same subject and object in one sentence for which Blank et al. (1992) reported no generalisation of the classification network.

In contrast to the previous experiments, now training of both the LRAAM and the classification network was performed in parallel. The LRAAM was trained to encode and decode the structures. At the same time the perceptron was trained to classify them and the errors for both tasks were propagated backwards through the architecture. The training of the combined networks was continued until the training examples were correctly classified.

For all classification tasks the architecture generalised with over 90% accuracy to unseen examples. Interestingly, good classification performance was achieved even with very poor decoding performance of the LRAAM. In one test case the classifier generalised the test set with 98.6% accuracy when the LRAAM decoded only 4.25% of the training set correctly. This effect is caused by the parallel training of the two networks. As Sperduti et al. (1995, p. 511) point out:

> "If no interaction at all is allowed, i.e., the LRAAM is trained first and then its weights are frozen, the reduced representations will be such that similar representations will correspond to similar structures, while if full interaction is allowed, i.e., the LRAAM and the classifier are trained simultaneously, the reduced representations will be such that structures in the same class will get very similar representations."

In the first case, the weights in the LRAAM are adjusted depending on the decoding error only. Therefore, after successful training the LRAAM is able to correctly decode the training examples. However, the resulting representations might not carry enough information for the classifier network to correctly classify the training set or even generalise to unseen examples. In the latter case, the representations for terms are developed under the influence of both the encoding/decoding and the classification task. The resulting representations are tuned towards the classification task, which will result in successful classification but probably unsuccessful decoding of the LRAAM. As the results show, despite the poor decoding performance of the LRAAM, the interleaved training of LRAAM and perceptron enables the architecture to accurately learn classifications that the decoupled architecture presented by Blank et al. (1992) was not able to perform.

3.2.2 Matching of Structures

Stolcke and Wu (1992) trained a 50-25-50 dual RAAM to encode binary trees with leaves taking on two different values: '0' representing a variable and '1' representing a non-variable element. These values were again encoded by a localist binary representation. The RAAM was trained to encode all possible tree structures with up to five leaves. Training was repeated until the RAAM perfectly encoded and decoded all training examples. A perceptron was then trained to perform the holistic matching or unification of the tree representations formed in the hidden layer of the RAAM. Matching was defined as follows:

1. A variable node '0' matches any tree. This tree is the result of the matching operation.

2. An element node '1' only matches an element node '1' which is the result of the matching operation.

Figure 3.3 shows a few examples of matching trees. The matching operation is represented by '∪'. 1001 training and 404 test examples for the perceptron were formed in

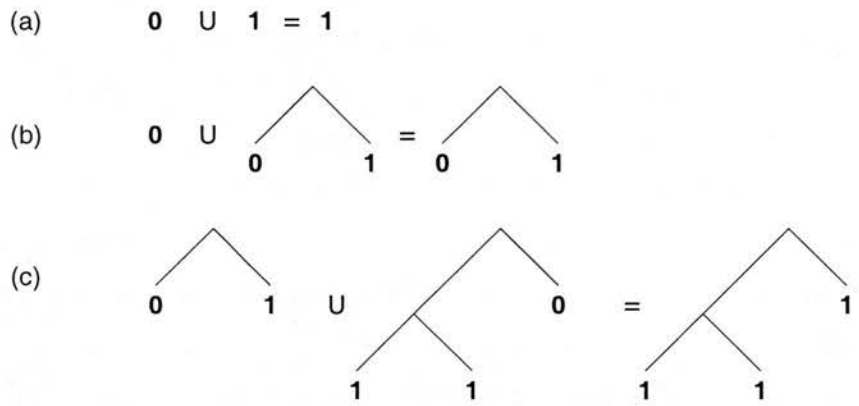

Figure 3.3: Matching of (a) variable and element, (b) variable and tree, and (c) two trees containing variables and elements. The picture is adapted from Stolcke and Wu (1992).

the hidden layer of the RAAM. Stolcke and Wu (1992) investigated two modes of training the two networks. Initially, RAAM and unification network were trained one after the other. In a second run, training of the two networks was performed in parallel so that the representations of trees developed under the influence of both the encoding in the RAAM and the matching task. In several tests this parallel architecture performed with 98.9% to 100% accuracy on the training and with between 90.5% and 100% on the test sets. The initial simpler training scenario, however, was not always successful. As Stolcke and Wu (1992, p. 6) observe, "[n]ot only does unification influence the representation under parallel training, but in fact, depending on initial conditions, we have sometimes only been able to achieve good unification performance with parallel training." As in Sperduti et al.'s classification experiment, the holistic operation could only be successfully learned when the representations formed in the hidden layer of the RAAM were modified to best fit the particular task they were applied to.

3.2.3 Transformation of Structures

Transformations with Level 1 Systematicity

Pollack (1988) and Chalmers (1990) were among the first to show that representations formed in the hidden layer of a RAAM can be transformed holistically. Already when introducing RAAMs, Pollack anticipated their capacity to learn distributed representations of hierarchical structures that can form the basis of fast inference and structural transformation engines. He observed that a pattern associator should be able to learn transformations of representations formed in a RAAM, assuming that it is possible to operate on these representations holistically.

Like the holistic classification and matching of structures, the transformation process starts with learning the representations of hierarchical structures by a RAAM and collecting the representations from the hidden layer of the RAAM. Then training examples, i.e., pairs of input and target structures, for the transformation are formed. These are presented to a transformation network (TN). In all experiments reported here a three-layer backpropagation network was used. To test for a successful transformation of structures the results collected from the output layer of the TN are fed into the hidden layer of the RAAM for decoding. The correct decoding of the presented structures indicates a successful transformation in the TN.

Pollack (1988) trained a TN with 16 input and output units and 8 hidden units on a number of inferences of the form

$$\text{IF } \mathit{Love}\,(X, Y) \text{ THEN } \mathit{Love}\,(Y, X)\,.$$

Representations for the relations $\mathit{Love}\,(X, Y)$ and $\mathit{Love}\,(Y, X)$ were formed in a ternary RAAM. After learning 12 inferences, the TN successfully generalised the transformational inference to 4 previously unseen examples.

Chalmers (1990) extended Pollack's work. He trained a ternary RAAM to encode syntactically simple active sentences like "John loves Helen" and their corresponding passive forms ("Helen is loved by John"). A TN was then trained to perform transformations from active to passive sentences and vice versa.

RAAM and TN were trained with 40 pairs of active and passive sentences. A test set of further 40 sentence pairs unknown to both the RAAM and the TN was then

presented. Only 65% of these sentences were generalised correctly. This result seemed to contradict the results reported by Pollack. Chalmers observed, however, that the poor generalisation did not reflect the inability of the TN to learn the transformation of the reduced representations. Rather, it was caused by a high generalisation error of the RAAM. When the test was repeated with examples unknown to the TN but known to the RAAM, the TN generalised correctly to all test examples. Moreover, Chalmers tested the generalisation capacity of the RAAM without transformation. The RAAM was trained with 80 randomly generated active and passive sentences and tested using 80 previously unseen examples. The RAAM generalised only 80% of the test sentences correctly.

This experiment reveals a major drawback of RAAMs. Once the RAAM has been trained to encode and decode a certain set of hierarchical structures, it poorly generalises to previously unseen structures. The same was observed by Blank et al. (1992) for sequential RAAMs. When the sequential RAAM described in Section 3.2.1 was trained on 100 simple sentences and its decoding was tested on 100 unseen examples without applying any holistic operation at all, it decoded only 80% of the test sentences correctly.

Blank et al. (1992) also investigated the holistic transformation of sentences encoded by a sequential RAAM. A TN was trained on 16 transformations of the form

$$X \textit{ chase } Y \implies Y \textit{ flee } X$$

where the sentences were constructed and encoded by a sequential RAAM as described in Section 3.2.1. The TN was first tested using 4 examples known to the RAAM. The results of the transformations were decoded by the RAAM with 100% accuracy. When the test was repeated with 4 sentences unknown to the RAAM, only 75% of the sentences were decoded correctly. Although both test sets were very small, they show again the good generalisation performance of the TN but a poorer performance of the RAAM.

In these experiments only transformations that require a level 1 systematicity were performed. All elements in the structures occurred in all their possible syntactic positions in the training sets[2]. Further, for all transformations a localist binary representation was used to encode elements. Chalmers, for example, used 13 binary units to

[2]Blank et al. (1992) did not actually report on the choice of sentences for the test and training set.

represent the words of the sentences. The first 6 units represented the part of speech of the word, the following 5 units represented the particular word, and the final 2 units were spare units only used for forming the distributed representation in the hidden layer of the RAAM. In order to decide which word the RAAM had decoded, the most highly activated of the part-of-speech units and the most highly activated of the word units were determined. The word was considered correctly decoded, if these units corresponded to the active units in the target word.

Transformations with Level 2 and Level 3 Systematicity

In a series of experiments Niklasson and colleagues investigated the holistic transformation of expressions in the domain of propositional logic. Niklasson and Sharkey (1997) first showed that holistic transformations which require level 1 systematicity can be learned for more complex embedded structures than the ones used by Pollack and Chalmers. A ternary RAAM was trained to encode/decode a set of 156 expressions. An expression was either an element p, q, r or a negation, conjunction, or disjunction with up to two embedded expressions such as, for example, $((\neg p) \vee (q \wedge r))$. A TN was trained to transform half of these expressions according to the axiom

$$expr_1 \rightarrow expr_2 \iff \neg\, expr_1 \vee expr_2\,.$$

The training set of the TN was designed such that every element occurred in all its allowed syntactic positions ensuring that only level 1 systematicity was required for the transformation. Note, however, that the transformation was performed in both directions. After training, the network was able to transform the training set and the remaining unseen expressions with 96% accuracy.

In a second experiment the TN was only trained on structures that did not contain the element p in a certain syntactic positions. A successful transformation now required processes that possess systematicity of level 2. Presented with the full test set, the TN transformed the expressions with 97.5% accuracy.

For both tasks a localist binary type-token representation for elements was used, similar to the one used by Chalmers. It is important to point out that using such an

However, there was no indication that elements had been left out of particular syntactic positions or that completely novel elements were introduced to the test set at any point.

input representation provides some knowledge about the syntactic structure of the hierarchical expressions that could be used in the structure-sensitive transformation process. By using a type-token representation it "is explicitly stated in the representation what role a certain object plays" (Niklasson and Sharkey, 1997, p.228). The structure sensitivity of the transformation might therefore be a result of the input representation to the RAAM. However, it can be shown that at least a low level of systematicity can be achieved using randomly chosen representations for elements in a RAAM. When the test for level 1 systematicity was repeated with a binary representation for elements containing a number of randomly chosen active units, the TN was able to generalise to unseen examples with 98% accuracy.

Finally, Niklasson and Sharkey presented some data indicating that a TN can be trained to perform a number of different holistic processes over the same reduced representations. The training and test sets of the TN as used so far were extended to include 12 additional structure pairs that could be transformed by four new transformational rules such as

$$\neg\,(expr_1 \wedge expr_2) \Longleftrightarrow \neg\, expr_1 \vee \neg\, expr_2\,.$$

In order to transform all structures correctly, the TN now had to be sensitive to different structural transformations of its inputs. After re-training, the TN was able to transform the test set with 98% accuracy.

Niklasson and van Gelder (1994a) demonstrated that a TN is able to perform transformations which require systematic processing of level 3. For such a degree of systematicity the TN has to generalise learned transformations to hierarchical structures containing entirely novel elements. A ternary RAAM was trained to encode expressions of the forms $x_1 \rightarrow x_2$ and $\neg\, x_1 \vee x_2$ where x_1 was a single element $x_1 \in \{p, q, r, s\}$ and x_2 was either a single element or an implication or disjunction of two elements. Training and test examples were then constructed for a transformation according to the axiom

$$x_1 \rightarrow x_2 \Longrightarrow \neg\, x_1 \vee x_2\,.$$

When the TN was trained on all structures not containing the element s in any position, it was able to perform the transformation of structures containing s without any errors.

The model used by Niklasson and van Gelder differed from the previous ones in

two ways. Firstly, distributed representations were used for elements. These were generated by a sequential RAAM according to the following scheme:

$$\begin{aligned} p^D &= [\ p\ [\ \text{Proposition}\ [\ \text{Symbol}\ \textit{nil}\ ]\ ]\ ] \\ \vee^D &= [\ \vee\ [\ \text{Connective}\ [\ \text{Symbol}\ \textit{nil}\ ]\ ]\ ]. \end{aligned}$$

The original atomic elements $\{p, q, r, s, \vee, \textit{nil}, \text{Proposition}\ldots\}$ were assigned orthogonal binary representations of 22 units, two of which were active in each pattern. Note that although the resulting representations are distributed, structures of the same type (i.e., Proposition or Connective) differed in only one element. Secondly, RAAM and TN were trained in parallel and errors occurring in the TN were also propagated to the RAAM. Thus, the reduced representations in the hidden layer of the RAAM were formed under the influence of both the encoding/decoding of the structures and their transformation in the TN.

Niklasson (1993) demonstrated level 3 systematicity for a combined architecture of RAAM and TN in a more complex domain. We analyse the results in detail here, because the domain will serve as a testbed for the transformation of HRRs presented in Chapter 4.

A combined architecture of a ternary RAAM and a three-layer backpropagation TN was trained to perform the transformations

$$\begin{aligned} (x_1\ op\ x_2) \rightarrow x_3 &\iff \neg\,(x_1\ op\ x_2) \vee x_3 \\ x_1 \rightarrow (x_2\ op\ x_3) &\iff \neg\, x_1 \vee (x_2\ op\ x_3) \end{aligned}$$

with $x_1, x_2, x_3 \in \{p, q, r, s\}$ and $op \in \{\rightarrow, \vee\}$. Training of both RAAM and TN was performed in parallel and a special bit was introduced in the input and output layer of the RAAM, indicating whether a represented expression was atomic or complex. The following localist binary representation with 40 units was used for representing elements:

$$\begin{aligned} p &= 1110000000000000000000000000\ldots \\ q &= 1001100000000000000000000000\ldots \\ r &= 1000011000000000000000000000\ldots \\ \vee &= 0000000001110000000000000000\ldots \end{aligned}$$

$$\begin{aligned} \rightarrow &= 0000000000001110000000000000\ldots \\ \neg &= 0000000000000001110000000000\ldots \\ nil &= 0000000000000000001110000000\ldots \end{aligned}$$

The test was performed with three different representations for the element s :

$$\begin{aligned} s &= 1000000000000000000000000000\ldots \\ s &= 0101010000000000000000000000\ldots \\ s &= 0010101000000000000000000000\ldots \end{aligned}$$

The architecture was first trained on all possible expressions not containing the element s. After successful training, it was only able to correctly transform and decode 50% of the expressions containing the element s. In order to investigate whether this high error was caused by the poor generalisation of the RAAM as previously reported, the RAAM part of the architecture was further trained to decode/encode all 512 possible expressions of the domain, including the expressions containing the element s. This additional training enabled the architecture to perfectly transform and decode all expressions, i.e., the TN successfully generalised the transformation to expressions containing a novel element.

Our Own Experiments

The architectures discussed so far show different degrees of systematic processing, but also employ different encoding mechanisms for the representation of elements and different training regimes and are thus not always directly comparable. We therefore replicated and modified Niklasson's experiment in order to investigate

1. the influence of the initial element representations on the generalisation performance of the system and

2. the influence of the particular architecture used, i.e., the parallel training of RAAM and TN and the special bit added to the representation that distinguished an element from a structured expression.

We implemented a ternary RAAM and a TN using the network simulator XERION. Our implementation differed from Niklasson's architecture in two ways. Firstly, TN and RAAM were trained independently. Secondly, we represented the element s by the pattern

$$s = 1000000110000000000000000000\ldots$$

which seems to us the most logical choice given the representations of all other elements. For this architecture we observed the following results.

Training Performance of the RAAM

The ternary RAAM was trained to encode/decode all 512 possible structures of the domain. After training, all decoded element representations were collected and tested against the target vectors. The Euclidean distance between the result of the decoding and all possible target vectors was used to determine the results of the transformation. 1.76% of all 3072 elements present in the structures were wrongly decoded. Errors occurred only for elements embedded in the lowest possible level in the hierarchical structures.

Training Performance of the TN

The TN was trained on the full example set to perform the holistic transformation of the structures. The fully trained RAAM was again used to decode the transformed structures. 6% of all elements were decoded incorrectly. 129 of the wrongly decoded elements were embedded in the lowest possible level in the hierarchies, the remaining 58 wrongly decoded elements were embedded in the second level.

Generalisation Performance of the RAAM

In order to test the generalisation performance of the RAAM it was re-trained using only structures not containing the element s. When it was presented with all possible structures, 14.3% of all elements were incorrectly decoded. The wrongly decoded elements included all 384 vectors supposed to represent the elements s and 55 (1.8%) other elements. This result confirms the findings observed earlier that once a RAAM is

trained on a particular set of examples, it generalises only poorly to previously unseen structures, in particular when they contain entirely novel elements. The RAAM on its own does not possess level 3 systematicity.

Generalisation Performance of the TN

In order to test the generalisation performance of the TN it was re-trained using only examples not containing the element s. All 512 structures were then presented to the TN and the results of all transformations where decoded by the fully trained RAAM. In contrast to Niklasson's reported 100% generalisation performance, 19.5% of all elements were now decoded wrongly. Given the very few decoding errors caused by the fully trained RAAM, most of the errors must have been caused by a wrong transformation. In fact, almost all result vectors supposed to represent the element s (11.75% of all elements) were decoded incorrectly. Interestingly, these vectors were decoded as

$$s' = 1000000000000000000000000000\ldots$$

which is the vector used as the first input representation for s in Niklasson's experiment.

Generalisation Performance of the RAAM and the TN

Finally, both RAAM and TN were trained using only expressions without the element s. Niklasson reported about 50% decoding errors in this case. However, in our system only 16.99% of all elements including all vectors representing the novel element were incorrectly decoded. That this error does not significantly differ from the generalisation errors of the RAAM and the TN alone can be explained by the fact that both networks performed wrongly on almost the same set of examples, namely on the structures containing the novel element. Niklasson's reported high generalisation error was presumably caused by the parallel training of the RAAM and TN. As discussed earlier, training both networks in parallel modifies the representations formed in the RAAM to best fit the holistic operation and not the decoding.

Performance of the RAAM and the TN without the Element/Structure Classifier

We repeated the set of experiments using a RAAM without the extra unit that distinguishes elements from composed structures. Niklasson (personal communication) observed a decreased performance of the TN in a parallel architecture when no extra unit is used. However, the results reported in Table 3.1 show that in our decoupled architecture the extra bit does not have any effect on the performance of either the RAAM or the TN. The errors observed do not significantly differ from the errors in our initial tests.

Test	Error
Training of RAAM	1.30 %
Training of TN	5.47 %
Generalisation of TN	16.34 %

Table 3.1: Performance of a decoupled RAAM and TN using no element/structure classifier in the representations.

Performance with Niklasson's Representation for the Element s

We further repeated the experiments using one of Niklasson's representations for the element s:

$$s = 0101010000000000000000000000\ldots$$

The 100% generalisation performance of Niklasson's architecture could not be replicated with a decoupled architecture. The results of our experiments are shown in Table 3.2. These results do not significantly differ from the results when using our initial representation for s. The element s was wrongly decoded 351 times (out of 384), 110 as p, 123 as q, and 118 times as r. The network did not generalise to the novel element.

Test	Error
Training of RAAM	0.00 %
Training of TN	5.66 %
Generalisation of TN	17.15 %

Table 3.2: Performance of a decoupled RAAM and TN using Niklasson's representation for the element s.

3.2.4 Discussion

The experiments presented in this Section address different tasks implemented as holistic operations on structures formed by RAAMs. Every experiment on its own gives convincing evidence that holistic operations such as the classification, unification, and transformation of hierarchical structures can be successfully performed on these representations. However, looking at the presented data in a comparative way reveals some interesting aspects of the nature of RAAM representations and their usefulness as basis for holistic operations. A successful holistic operation on RAAM structures and the achievable complexity of this operation strongly depends on

1. the initial representation of structure elements and
2. the particular architecture and the training regime chosen.

The experiments regarding holistic transformations in particular show the significance of the initial element representation for the generalisation capacity of a network learning the holistic operation. Some networks discussed performed generalisations that require level 1 systematicity. This simple generalisation can be performed using randomly chosen representations of elements. However, for all transformations that require a higher level of systematicity, representations for elements were chosen on the basis of similarity. One model requiring level 3 systematicity (Niklasson, 1993) employed a type-token representation for elements. Within this framework the representation of the novel element was carefully chosen such that it captured information about all elements of the same type. We would argue that the representation chosen by Niklasson for the unseen element s

$$s = 1000000000000000000000000000\ldots$$

even represented the type of the elements p, q, and r, say the type *proposition*, rather than a new instance of that type. Recall that in our own experiments the element s was often wrongly decoded as this pattern. Phillips (1998) came to a similar conclusion, suggesting that the generalisation was possible without s in any training examples, because the first unit in *all* propositions was set to 1. With this encoding, Phillips argues, even the novel constituent s received training. It would be interesting to see how well Niklasson's architecture performs on elements for propositions that do not share any activations of units or on the apparently more logical representation

$$s = 1000000110000000000000000000\ldots$$

The second model that possessed level 3 systematicity (Niklasson and van Gelder, 1994a) employed a distributed representation for elements which was formed in the hidden layer of a sequential RAAM. As observed, for example, by Pollack (1989) and Niklasson and Bodén (1997) representations formed in a RAAM are similar to each other if the syntactic structure of the compositions and the elements they contain are similar. The way the element representations were formed in Niklasson and van Gelder's model suggest that again representations for elements of the same type were very similar. The generalisation of the model was based on this similarity.

The architectures used in the reported experiments fall into two groups. In one group the RAAM is first trained on the encoding task and only after a good encoding and decoding performance is achieved, the TN, classification network, or unification network is trained. In this way the representations of hierarchical structures are developed independently from the holistic operation they are applied to. In the second group, training of both networks is performed in parallel. In this case the error observed when learning the holistic operation affects the weight changes in both networks and the representations developed in the hidden layer of the RAAM are shaped by both the encoding process and the holistic operation. Both training regimes enable connectionist networks to learn holistic operations over hierarchical structures to some degree. In fact, Niklasson (personal communication) expected only a slight difference in performance between a decoupled and a combined architecture learning the same task. Looking at the experiments presented here shows, however, that architectures using independent training only seem to master relatively simple tasks: detection of elements

in structures, transformations with level 1 systematicity. In contrast, architectures using parallel training of both networks are able to learn more demanding tasks: tree unification, transformations that require level 3 systematicity. While the first architectures prove the suitability of RAAM representations for holistic processing in general, they fail to perform complex operations likely to be required in high-level cognitive processes. Such demanding processes seem to ask for RAAM-based representations that are optimised to fit the particular task at hand.

We regard it as one of the major strengths of RAAMs that the representations for hierarchical structures are *learned* as opposed to *constructed.* Although it is difficult to directly compare representations formed by different mechanisms, we would expect RAAMs to make better use of the representational space than constructing methods. The relatively low dimensionality of RAAM vectors required to encode complex structures could be taken as empirical evidence for this (Plate, 1997b). The optimal use of the representational space can further be enhanced by training the RAAM not only to encode and decode hierarchical structures but also to distinguish atomic elements from already composed objects. This is achieved by inducing an extra bit in the input and output layer of the RAAM functioning as an indicator for atomic elements. Indeed, Niklasson (1999) observes that using this extra bit results in a more effective use of the hidden activation space of the RAAM and in a clearer structure of the representational space. Surprisingly, though, our experiments indicate that in a decoupled architecture this clearer structure does not result in a better performance of either network. Using the extra bit does, however, simplify the process of decoding transformed structures.

One of the drawbacks of using RAAMs to form representations of hierarchical structures is their poor generalisation capacity. Several of the experiments presented above confirm that once representations have been formed in the hidden layer of a RAAM, it only generalises poorly to the encoding and decoding of unseen examples. Thus, the extension of a domain to new hierarchical structures or even new elements will often require the re-training of the RAAM.

The decoding and generalisation performance of RAAMs is particularly poor when they are trained in parallel with the network learning the holistic operation. This is problematic for the holistic transformation of structures which, in contrast to classification, requires the decoding of the result structures in order to test for the correct

performance of the transformation network. It is therefore desirable to perform the training of RAAM and TN independently. This method does not seem to yield satisfactory results for relatively difficult transformation tasks that require a high degree of systematicity, however.

Some experiments on the holistic transformation of structures suggest that a TN is able to apply more than one transformation to hierarchical structures at once. Niklasson and van Gelder (1994a) and Niklasson (1993), for example, trained the TN to perform not only a single transformation but also its reverse. Depending on whether the input structure represented a disjunction of expressions or an implication, the appropriate transformation was performed turning the structure into the logically equivalent implication or disjunction, respectively. Niklasson and Sharkey (1997) successfully trained a TN to perform 6 different transformations at once. It has to be noted, however, that the number of test and training examples for 4 out of the 6 transformations was very small and it remains unclear whether these particular transformations have truly been learned and can be generalised to unknown structures or whether these few cases have simply been remembered by the network.

3.3 Holistic Operations on HRRs, Tensor Products, and BSCs

3.3.1 Matching of HRR structures

We have seen in Chapter 2 that the two basic operations for HRR, superposition by vector addition and binding by circular convolution, preserve the similarity of vectors in a complementary fashion. Superposition preserves unstructured similarity. The result of the superposition of two vectors is similar to both vectors. Circular convolution, in contrast, is randomising but it preserves structured similarity: The circular convolution $\mathrm{A} \circledast \mathrm{B}$ is similar to $\mathrm{C} \circledast \mathrm{D}$ to the extend that A is similar to C and B is similar to D. Given these similarity preserving properties of the operations, the dot product[3] of HRRs reflects the similarity of the hierarchical structures they represent. The dot prod-

[3]Recall that HRR vectors are usually of expected length 1. Thus, the dot product of two HRR vectors does not need to be normalised in order to reflect their similarity.

uct thus implements a holistic operation that estimates the structural match of HRRs. Plate (1997b) demonstrated this in the following domain. A number of episodes were constructed from role-filler bindings involving the elements *Jane, John, Fido, Spot, Felix, Mort, bite, flee,* and *cause*. Similarities between some of the elements were expressed by convolving the randomly chosen initial element vectors with appropriate types. For example, the elements *Jane* and *John,* belonging to the same type *person,* were represented by the vectors

$\mathbf{Jane} = \mathbf{person} \circledast \mathbf{jane\_id}$ and
$\mathbf{John} = \mathbf{person} \circledast \mathbf{john\_id}$.

The elements *Fido* and *Spot* are of type *dog, Felix* is of type *cat,* and *Mort* is a *mouse.* The representation of the episode

E1: *Spot bit Jane, causing Jane to flee from Spot.*

was then matched by the dot product against three other episodes with different degrees of similarity to E1. Table 3.3 shows the results of the matching.

Episode	Dot product with E1
E2: *Fido bit John, causing John to flee from Fido.*	0.81
E3: *Felix bit Mort, causing Mort to flee from Felix.*	0.61
E4: *Mort bit Felix, causing Mort to flee from Felix.*	0.39

Table 3.3: Episodes matched against E1 and the results of the holistic matching operation. The results are averaged over 100 tests with different randomly chosen element vectors.

The episodes E2 and E3 belong to the same class of structures as E1. The episode E2 is literally similar to E1. The two episodes share the syntactic structure and the roles, and the fillers belong to the same types. As expected, the dot product of the corresponding HRRs is very high. E3 is an analogy to E1, i.e., the syntactic structure and the roles of the two episodes are the same, but the fillers belong to different types. Consequently, the representations of the two episodes are less similar, which is indeed reflected in the result of the holistic matching. Finally, E4 is structurally different from E1 and has different fillers. As one would expect, the dot product of the two representations is considerably smaller than the dot product in the previous two cases.

This small example shows that the dot product is a simple and efficient implementation of a holistic operation that provides an estimation of the similarity of hierarchical structures represented by HRRs. Plate (1998a) employs this holistic operation for example in a model for analogical retrieval. It should be noted, however, that the form and the degree of similarity estimated by the dot product strongly depends on the way the HRR structures are encoded in the first place.

3.3.2 Transformations of Tensor Products, HRRs, and BSCs

Transformations of Tensor Products

Legendre et al. (1991) investigated the holistic transformation of hierarchical structures represented by Tensor Products. We will first explain the general mechanism on a simple example. Representations of parse trees for simple English sentences can be constructed from filler vectors representing the words of the sentences and two role vectors $\mathbf{r_0}$ and $\mathbf{r_1}$ representing the left child and the right child of a node in the tree, respectively. Other positions in a tree are represented by role vectors composed recursively from $\mathbf{r_0}$ and $\mathbf{r_1}$. The role vector $\mathbf{r_{00}} = \mathbf{r_0} \otimes \mathbf{r_0}$ represents the left child of the left child of a node, the role vector $\mathbf{r_{01}} = \mathbf{r_0} \otimes \mathbf{r_1}$ represents the left child of the right child of a node, and so on. Using this mechanism we can represent, for example, the tree in Figure 3.4a by the tensor

$$\textbf{John\_loves\_Helen} = (\mathbf{r_0} \otimes \textbf{John} + \mathbf{r_{01}} \otimes \textbf{loves} + \mathbf{r_{11}} \otimes \textbf{Helen}) .$$

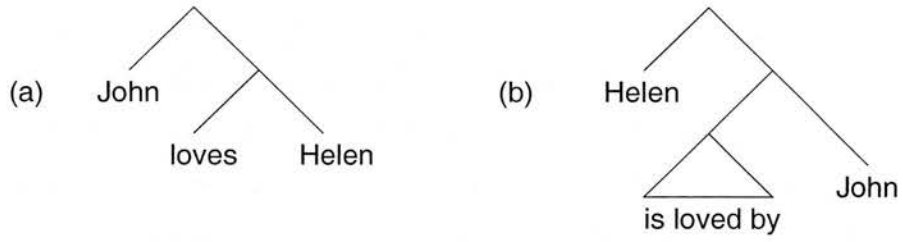

Figure 3.4: Tree structures representing (a) a simple active sentence and (b) its passive counterpart.

Note that some tensors in this representation, e.g., $\mathbf{r_0} \otimes \textbf{John}$ and $\mathbf{r_{01}} \otimes \textbf{loves}$ are of different rank. Therefore, a special operation adding tensors of different ranks needs

to be implemented.

Tensors constructed in this way can be transformed holistically by a single matrix-vector multiplication. We can construct a transformation matrix that when combined with a tensor representing a structure performs all modifications on the structure necessary for the full transformation. For example, in order to transform the representation for the tree in Figure 3.4a into the representation for tree in Figure 3.4b we have to unbind **John** and **Helen** using unbinding vectors $\mathbf{u_0}$ and $\mathbf{u_{11}}$, newly bind **Helen** and **John** to $\mathbf{r_0}$ and $\mathbf{r_{11}}$, respectively, and change the verb into its passive form. Tensors for these operations can be constructed and superimposed to a matrix that transforms the active sentence into its passive counterpart. Note that although the transformation consists of several steps, it is performed as a single operation once the transformation matrix is constructed and does not require any explicit decomposition of the structured representations. In can therefore be regard as a holistic operation. However, the unbinding vectors $\mathbf{u_0}$ and $\mathbf{u_{11}}$ corresponding to the roles $\mathbf{r_0}$ and $\mathbf{r_{11}}$, respectively, have to be calculated in advance. Alternatively, the role vectors can be used to unbind the fillers which, however, increases the amount of noise in the transformation result.

Legendre et al. (1991) implemented a two-layer connectionist network performing transformations similar to the example above. A network was designed to transform tree representations for syntactic parses of English sentences into tree representations for predicate-calculus expressions standing for the meaning of the sentences. Both active and passive sentences were considered, so the network had to perform the two transformations illustrated in Figure 3.5.

Two transformation matrices were implemented in the network as separate weight matrices connecting input and output layer and performing the two different transformations. The input layer was presented with the tensor of an active or passive sentence and the tensor representation for the predicate-calculus expression was produced in the output layer of the network. A marker in the input representation indicated whether an input was an active or a passive sentence and the connections used to calculate the output were chosen accordingly. The network so constructed was able to accurately process arbitrary input sentences of either type up to tree depth 4. However, it is important to be clear that, although the transformations were performed by a connectionist network, they were not learned from examples. Rather the network was *designed* to

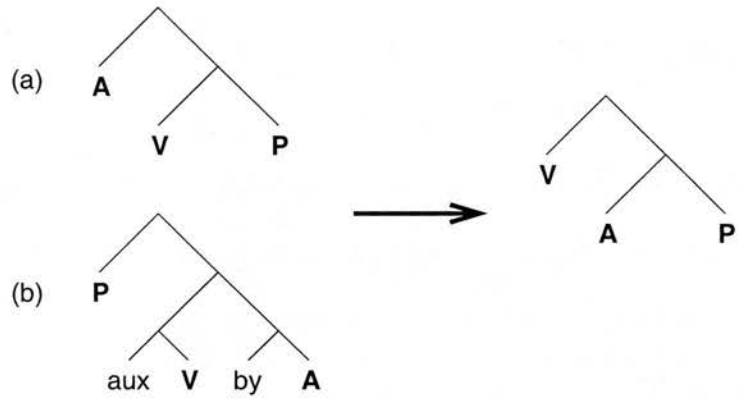

Figure 3.5: Tree representation for (a) active sentences and (b) passive sentences that are transformed into the tree representation for the predicate-calculus expression $\mathbf{V}(\mathbf{A}, \mathbf{P})$. $\mathbf{V}, \mathbf{A}, \mathbf{P}$ stand for *verb, agent,* and *patient*, respectively.

implement the weight matrices after their construction from the single operations that the transformations were comprised of.

Transformation of HRRs

Plate (1997b) showed that similar holistic transformations can be performed with HRR. A structure $\mathbf{A}$ is transformed into a structure $\mathbf{B}$ by constructing a transformation vector $\mathbf{T}$ such that $\mathbf{A} \circledast \mathbf{T} = \mathbf{B}$. Clearly, $\mathbf{T}$ is unique for any pair $(\mathbf{A}, \mathbf{B})$. However, given the noise tolerance of HRRs, a transformation can also be performed on a whole class of structures. For two classes of structures $\{\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n\}$ and $\{\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_n\}$ a transformation vector $\mathbf{T}$ can be constructed such that $\mathbf{A}_i \circledast \mathbf{T} = \tilde{\mathbf{B}}_i$ for all $i = 1 \ldots n$ where $\tilde{\mathbf{B}}_i = \mathbf{B}_i + \textit{noise}$.

Plate (1997b) applied this technique to the domain used by Niklasson and van Gelder (1994a) discussed in Section 3.2.3. A transformation vector was constructed for transformations of the form

$$x_1 \rightarrow x_2 \Longrightarrow \neg x_1 \vee x_2$$

where x_1 was a single element $x_1 \in \{p, q, r, s\}$ and x_2 was either a single element $\{p, q, r, s\}$ or an implication or disjunction of two elements. Representations for structures were constructed using the following scheme:

$\mathbf{X\_impl\_Y} = (\mathbf{impl} + \mathbf{ante} \circledast \mathbf{X} + \mathbf{cons} \circledast \mathbf{Y})$

$\mathbf{not\_X\_or\_Y} = (\mathbf{disj} + \mathbf{neg} \circledast \mathbf{X} + \mathbf{pos} \circledast \mathbf{Y})$

The constructed transformation vector was of the from

$\mathbf{T}_1 = (\mathbf{impl}' \circledast \mathbf{disj}) + (\mathbf{ante}' \circledast \mathbf{neg}) + (\mathbf{cons}' \circledast \mathbf{pos})$

$\mathbf{T}_2 = (\mathbf{disj}' \circledast \mathbf{impl}) + (\mathbf{neg}' \circledast \mathbf{ante}) + (\mathbf{pos}' \circledast \mathbf{cons})$

$\mathbf{T} = \mathbf{T}_1 + \mathbf{T}_2$

Both structures and transformation vectors were normalised during construction. The transformation accuracy of the constructed transformation vector was tested for HRR vectors of various dimensions by decoding the results of the transformations into their elements. For 4096-dimensional vectors an average decoding error of 1.2 elements (out of 4800) occurred over 10 experiments. The performance significantly degraded for lower-dimensional vectors.

Our Own Experiment

We applied the method to the more complex domain used by Niklasson (1993) discussed in Section 3.2.3. In order to provide more test structures, the domain was extended by the element t. The task in this domain was to perform the transformations

$$(x_1 \ op \ x_2) \rightarrow x_3 \iff \neg (x_1 \ op \ x_2) \vee x_3$$
$$x_1 \rightarrow (x_2 \ op \ x_3) \iff \neg x_1 \vee (x_2 \ op \ x_3)$$

with $x_1, x_2, x_3 \in \{p, q, r, s, t\}$ and $op \in \{\rightarrow, \vee\}$. All original elements of the domain p, q, r, s, t, *not*, *impl*, and *disj* and the roles *first*, *second*, *ante*, and *cons* were represented by randomly generated HRR vectors. For reasons of comparison we used the mechanisms proposed by Plate (1994) for the generation of element vectors and the clean-up of decoded results in this and all following simulations. Element vectors were generated using the vector calculator *vcalc0.9* (Plate, 1998b) and a list of all element vectors was stored. When cleaning up a decoded element the dot products of the retrieved element with all stored vectors were computed in order to find the closest match. All vector operations other than the vector generation were implemented in MATLAB.

For the holistic transformation above, 1000 different structures were formed according to the following three rules:

$$\mathbf{not\_X} = (\mathbf{not} \circledast \mathbf{X})$$

$$\mathbf{X\_impl\_Y} = (\mathbf{op} \circledast \mathbf{impl} + \mathbf{ante} \circledast \mathbf{X} + \mathbf{cons} \circledast \mathbf{Y})$$

$$\mathbf{not\_X\_or\_Y} = (\mathbf{op} \circledast \mathbf{disj} + \mathbf{first} \circledast \mathbf{not\_X} + \mathbf{second} \circledast \mathbf{Y})$$

with **X** and **Y** being either single elements or disjunctions or implications of two elements. A so constructed implication can be changed into its logically equivalent disjunction by a transformation vector that is comprised of the following three operations:

1. The top level implication of the structure has to be changed into a disjunction.
2. The antecedent of the implication has to be negated and becomes the first element of the disjunction.
3. The consequent of the implication becomes the second element of the disjunction.

For the reverse transformation, the reverse operations have to be performed. We constructed the transformation vector $\mathbf{T}$ for both transformations as follows:

$$\mathbf{T}_1 = (\mathbf{impl}' \circledast \mathbf{disj}) + (\mathbf{ante}' \circledast \mathbf{not} \circledast \mathbf{first}) + (\mathbf{cons}' \circledast \mathbf{second})$$

$$\mathbf{T}_2 = (\mathbf{disj}' \circledast \mathbf{impl}) + (\mathbf{first}' \circledast \mathbf{not}' \circledast \mathbf{ante}) + (\mathbf{second}' \circledast \mathbf{cons})$$

$$\mathbf{T} = \mathbf{T}_1 + \mathbf{T}_2$$

Vector dimensionality	Error
512	55.74 %
1024	96.50 %
4096	100.00 %

Table 3.4: Error when a constructed transformation vector was applied to structures represented by HRRs of different dimensionality.

After applying the transformation vector to a structure, the result was decoded into its elements. A transformation was considered correct if the result was completely

decoded into all elements it consisted of. We tested the transformations using vectors of different dimensionality with the results presented in Table 3.4. As in Plate's simulation, the transformations were performed correctly when 4096-dimensional HRR vectors were used, but performance degraded for lower-dimensional vectors.

Transformation of BSCs

The technique used for the holistic transformation of structures encoded in BSC is very similar to the construction of a transformation vector for HRR structures. This was illustrated by Kanerva (1998). The relational structures *'A is mother of B'* and *'A is parent of B'* can be encoded in BSC as

$$\begin{aligned} \mathbf{m_{AB}} &= (\mathbf{m} + \mathbf{m_1} \otimes \mathbf{A} + \mathbf{m_2} \otimes \mathbf{B}) \\ \mathbf{p_{AB}} &= (\mathbf{p} + \mathbf{p_1} \otimes \mathbf{A} + \mathbf{p_2} \otimes \mathbf{B}) \end{aligned}$$

where $\mathbf{m}$ and $\mathbf{p}$ encode the names of the relations and $\mathbf{m_1}, \mathbf{m_2}, \mathbf{p_1}$, and $\mathbf{p_2}$ encode the roles in the structures. Note that in this Section we use the symbol $\otimes$ for the operator XOR and not for the tensor product of two vectors. Recall that XOR is its own inverse operation. A vector transforming $\mathbf{m_{AB}}$ into $\mathbf{p_{AB}}$ can therefore be constructed as

$$\mathbf{T_{AB}} = (\mathbf{m_{AB}} \otimes \mathbf{p_{AB}})$$

because

$$\mathbf{m_{AB}} \otimes (\mathbf{m_{AB}} \otimes \mathbf{p_{AB}}) = \mathbf{p_{AB}}.$$

Kanerva (1998) observes that the superposition of so constructed transformation vectors generalises to unseen relational structures. For example, the vector

$$\mathbf{T} = \mathbf{T_{AB}} + \mathbf{T_{BC}} + \mathbf{T_{CD}}$$

correctly transforms the structure $\mathbf{m_{UV}}$ into the structure $\mathbf{p_{UV}}$. Note that for the superposition of the three transformation vectors not the normalised sum but the actual sum of the vectors is used. This is necessary because the normalisation discards too much information when superimposing a number of vectors (Kanerva, 1998). This

superposition operation, however, puts the result outside the range of binary vectors and bit-wise XOR can no longer be applied. Kanerva addresses this problem by using vectors with elements 1 and -1 instead of binary vectors and vector multiplication ($\times$) instead of bit-wise XOR as binding operation.

An analysis of the individual transformation vectors $\mathbf{T_{AB}}, \mathbf{T_{BC}}, \mathbf{T_{CD}} \ldots$ reveals that they are all of the form

$$\mathbf{m} \times \mathbf{p} + \mathbf{m_1} \times \mathbf{p_1} + \mathbf{m_2} \times \mathbf{p_2} + \mathit{noise}\,.$$

Thus, the vector

$$\mathbf{T^*} = \mathbf{m} \times \mathbf{p} + \mathbf{m_1} \times \mathbf{p_1} + \mathbf{m_2} \times \mathbf{p_2}$$

performs the transformation of the class of structures representing *is-mother-of* relations into a class of structures representing *is-parent-of* relations. Multiplying a structure of the first class with $\mathbf{T^*}$ replaces the structure name $\mathbf{m}$ with the structure name $\mathbf{p}$, unbinds the filler from role $\mathbf{m_1}$ and binds it to the role $\mathbf{p_1}$, and unbinds the filler from role $\mathbf{m_2}$ and binds it to the role $\mathbf{p_2}$. Like for the transformation of HRR structures, the transformation vector is constructed from the individual modifications necessary to transform a whole structure.

3.3.3 Discussion

The results presented in this Section show that structures represented by Tensor Products, HRRs, and BSCs can be operated on holistically. The similarity preserving properties of the binding and superposition operations used for the encoding of HRR structures allow for the fast and efficient estimation of their similarity in a single step. We would expect similar operations to work for Tensor Products and BSCs. All three representational schemes further support the holistic transformation of hierarchical structures. The main difference between the operations used here and the holistic transformation of RAAM structures is that the latter can be learned from examples while the holistic transformations of tensors, HRRs, and BSC are constructed by hand. Kanerva (1998) argues that the method of superimposing transformation vectors for single structures to a transformation vector for a class of structures can be regarded as learning the transformation from examples. We agree with Kanerva that in this process

knowledge about the transformation of a few structures is generalised to the transformation of a whole class of structures. The process of generating the transformation vectors for the example structures, however, is still one of construction and requires detailed information about the way the representations for the structures are formed and about the specific modifications of the structures that are required to perform the whole transformation. To our best knowledge no method exists so far that truly learns the holistic transformation of HRRs and similar representations from examples. We regard the learnability of representations and operations over them as an important issue, however. The work presented in the remainder of this thesis was therefore focussed on developing methods that learn holistic operations on structures represented by HRR or similar representations from examples.

We have seen in Section 3.2.3 that the noise tolerance of RAAMs permits learning the transformation of a number of different, although related, classes of structures at once. The same can be achieved by the superposition of vectors representing different HRR transformations. In our example, two transformation vectors $\mathbf{T}_1$ and $\mathbf{T}_2$ (cf. Section 3.3.2) transform an implication into a disjunction and a disjunction into an implication, respectively. Applying the superimposed vector $\mathbf{T} = \mathbf{T}_1 + \mathbf{T}_2$ to a structure of either class increases the amount of noise in the result of the transformation. However, as our simulation shows the result is still recognisable if structures and transformation vectors are represented by high-dimensional vectors. Note that the superimposed transformation vectors are related in that they represent a single transformation and its exact reverse. However, we expected the superposition of transformation vectors also to be possible for a number of different unrelated transformations. The conditions for the transformation of different classes of structures in parallel and the possibility to learn such transformations from examples will be investigated in Chapter 6.

The influence of the noise on the result of a HRR transformation strongly depend on the dimension of the vectors representing the structures. However, the recognisability of a transformed structure is also influenced by the actual modifications the transformation performs on the input structures. Plate (1994) observes that it is problematic to construct transformations that change structures only slightly and require to leave large parts of the input structures unchanged. The reason for this can be seen in

the following example. Suppose we want to transform the structure

$$\mathbf{S_I} = \mathbf{r}_1 \circledast \mathbf{X} + \mathbf{r}_2 \circledast \mathbf{Y}$$

into the structure

$$\mathbf{S_O} = \mathbf{r}_1 \circledast \mathbf{A} + \mathbf{r}_2 \circledast \mathbf{B}\,.$$

This transformation can be performed using the transformation vector

$$\mathbf{T} = \mathbf{X}' \circledast \mathbf{A} + \mathbf{Y}' \circledast \mathbf{B}\,,$$

because

$$\begin{aligned}\mathbf{S_I} \circledast \mathbf{T} &= \underbrace{(\mathbf{r}_1 \circledast \mathbf{X} \circledast \mathbf{X}' \circledast \mathbf{A})}_{\approx \mathbf{r}_1 \circledast \mathbf{A}} + \underbrace{(\mathbf{r}_2 \circledast \mathbf{Y} \circledast \mathbf{Y}' \circledast \mathbf{B})}_{\approx \mathbf{r}_2 \circledast \mathbf{B}} + \\ &\quad (\mathbf{r}_1 \circledast \mathbf{X} \circledast \mathbf{Y}' \circledast \mathbf{B}) + (\mathbf{r}_2 \circledast \mathbf{Y} \circledast \mathbf{X}' \circledast \mathbf{A}) \\ &= \mathbf{S_O} + \textit{noise}\,.\end{aligned}$$

Now consider the transformation of

$$\mathbf{S_I} = \mathbf{r}_1 \circledast \mathbf{X} + \mathbf{r}_2 \circledast \mathbf{Y}$$

into the structure

$$\hat{\mathbf{S}}_\mathbf{O} = \mathbf{r}_1 \circledast \mathbf{A} + \mathbf{r}_2 \circledast \mathbf{Y}\,.$$

Here we only want to replace $\mathbf{X}$ by $\mathbf{A}$ but keep $\mathbf{Y}$ bound to $\mathbf{r}_2$. The first choice for the transformation vector would be

$$\hat{\mathbf{T}} = \mathbf{X}' \circledast \mathbf{A}\,.$$

However, applying this transformation vector to $\mathbf{S_I}$ results in

$$\begin{aligned}\mathbf{S_I} \circledast \hat{\mathbf{T}} &= \underbrace{(\mathbf{r}_1 \circledast \mathbf{X} \circledast \mathbf{X}' \circledast \mathbf{A})}_{\approx \mathbf{r}_1 \circledast \mathbf{A}} + (\mathbf{r}_2 \circledast \mathbf{Y} \circledast \mathbf{X}' \circledast \mathbf{A}) \\ &= \mathbf{r}_1 \circledast \mathbf{A} + \textit{noise}\,.\end{aligned}$$

The result of the transformation is not the desired structure $\hat{\mathbf{S}}_\mathbf{O}$. In the process of the transformation the term $\mathbf{r}_2 \circledast \mathbf{Y}$ which was supposed to stay unchanged in the

structure was lost. Plate (1994) observes that in order to keep a term unchanged as part of a result structure, we have to include the identity vector $\mathbf{I} = (1, 0, 0, \ldots)$ into the transformation vector. In our example, the transformation vector should therefore be:

$$\hat{\mathbf{T}} = \mathbf{X}' \circledast \mathbf{A} + \mathbf{I}.$$

The result of the transformation now includes a full copy of the vector $\mathbf{S_I}$ with the term $\mathbf{r}_2 \circledast \mathbf{Y}$ unchanged:

$$\mathbf{S_I} \circledast \hat{\mathbf{T}} = \mathbf{r}_1 \circledast \mathbf{A} + \mathbf{r}_1 \circledast \mathbf{X} + \mathbf{r}_2 \circledast \mathbf{Y} + \textit{noise}.$$

However, this result of the transformation is ambiguous, because two fillers are bound to the role $\mathbf{r}_1$. We would therefore expect the correct filler not to be easily identifiable.

For a simple test of this hypothesis we applied the transformation vector $\hat{\mathbf{T}}$ to 46 different 4096-dimensional vectors of the form

$$\mathbf{S}_i = \mathbf{r}_1 \circledast \mathbf{X} + \mathbf{r}_2 \circledast \mathbf{Y}_i \qquad \text{for} \qquad i = 1 \ldots 46.$$

After the transformations, the fillers of the result structures were decoded by convolving the result structures with $\mathbf{r}_1'$ and $\mathbf{r}_2'$, respectively. While the filler bound to $\mathbf{r}_2$ was correctly decoded as $\mathbf{Y}_i$ in all cases, the filler bound to $\mathbf{r}_1$ was 37 times wrongly decoded as $\mathbf{X}$ and only 9 times correctly as $\mathbf{A}$. Plate (1994, p.151) maintains that "the only way to solve the problem is to do transformations [of parts of the structure] separately and clean-up intermediate results." However, this kind of transformation would no longer be a fully holistic operation. We will see in Chapter 4 that this problem can also be solved by learning the holistic transformation from examples.

3.4 Summary

In this Chapter we investigated a number of systematic structure-sensitive processes over connectionist representations of hierarchical structures. These processes differ from structure-sensitive processes in symbolic systems quite significantly, because of the fundamentally different nature of the representations they operate on. We could not put the difference into better words than Niklasson and Sharkey (1997) who observe that

> "[i]n a classical symbol manipulating system, the ability to define structure sensitive operations is due to the fact that the representations themselves carry syntactic information. This means that systematicity heavily depends on the ability to decompose the structured representations in search for certain constituents.
>
> In a connectionist system, the ability to define structure sensitive operations is ... due to the fact that the superpositional representations carry structural information without themselves being syntactically structured. This means that the connectionist type of systematicity heavily depends on the ability to define holistic operations on superpositional representations."

Representations formed in the hidden layers of RAAMs can form the basis of holistic operations such as the classification, the structural mapping, and the transformation of structures. The operations required are learned from examples by simple perceptrons or three-layer feedforward connectionist networks. Processing in these networks is highly systematic as the holistic operations are not only learned but also generalised to unseen examples, some even containing completely novel structure constituents. The experimental results show, however, that complex and demanding holistic processes require an interleaved acquisition of the structure representations and the actual holistic operation. We were, for example, not able to replicate Niklasson's transformation which generalised with level 3 systematicity in a decoupled architecture. The parallel training of the RAAM and the network performing the holistic operation results in structure representations that are optimised with respect to the holistic operation applied. Further, generalisations of holistic operations to novel elements in structures, i.e., processes with level 3 systematicity, require initial representations for elements that express the similarity of elements belonging to the same type. It is often assumed that generalisations to previously unknown elements in a structure can only be based on the similarity of these elements to already existing ones (e.g. Niklasson and van Gelder (1994b)). We will see in the next Chapter, however, that this does not always have to be the case.

Holistic processes such as structural mappings and transformations can also be defined over Tensor Products, HRRs, and BSCs. Our analysis here focussed on holistic

transformation processes. The main difference between these transformation processes and the transformations of RAAM structures and, we believe, the main drawback of the former is that they are constructed rather than learned from examples. The construction process requires detailed knowledge about the syntactic structure of the objects represented and, for complex hierarchical structures, can be complicated and sometimes tedious. Further, depending on the actual modifications of the structures necessary for the correct transformation, it is not always possible to construct a transformation vector that transforms a whole class of structures. In the following Chapter we will present two new methods of learning the holistic transformation of HRR structures and discuss how these methods overcome the difficulties observed here.

Chapter 4

Learning the Holistic Transformation of HRRs

We have seen in Chapter 3 that transformations of structures formed in the hidden layer of a RAAM can be learned from examples. In contrast, structures represented by Holographic Reduced Representation were transformed using a constructed transformation vector. In order to construct the transformation vector, the role vectors of the structures and the way these vectors are combined have to be known in advance, which is not a desirable situation. Moreover, the construction process becomes complicated as the input structures grow in size and complexity. For some classes of structures no accurate transformation vector can be constructed at all.

In this Chapter we present different methods of transforming HRR structures that do not rely on the construction of a transformation vector by hand. We first show how a single structure is transformed by the approximate and the exact inverse of circular convolution and then derive two methods for learning the holistic transformation of classes of structures from examples. We test the generalisation performances of the learned transformation vectors and compare them with results observed for the construction method. The work presented in this Chapter is in large parts published in Neumann (2000a) and Neumann (2000b).

4.1 Transformation of a Single Structure

In order to calculate the transformation vector for a single pair of structures we can use the approximate and the exact inverse operation to circular convolution. For a given pair of n-dimensional HRR structure vectors $\{\mathbf{A}, \mathbf{B}\}$ we want to find a transformation vector $\mathbf{T} = (t_1, \ldots, t_n)$ such that $\mathbf{A} = \mathbf{B} \circledast \mathbf{T}$. Given the distribution $\mathrm{N}(0, 1/n)$ of the randomly chosen vector elements of $\mathbf{A}$ and $\mathbf{B}$, a first approach is to use circular correlation to calculate $\mathbf{T}$ (cf. Equation 2.11):

$$\mathbf{A} = \mathbf{B} \circledast \mathbf{T} \tag{4.1}$$

$$\mathbf{B} \oplus \mathbf{A} = \tilde{\mathbf{T}}. \tag{4.2}$$

Since circular correlation is only an approximate inverse to circular convolution, $\tilde{\mathbf{T}}$ is only an approximation to the exact transformation vector $\mathbf{T}$. However, using Equation (2.19) we can calculate the exact inverse operation from the circulant matrix $\mathrm{circ}(\mathbf{B})$ which results in the exact transformation vector:

$$\mathbf{A} = \mathrm{circ}(\mathbf{B})\,\mathbf{T} \tag{4.3}$$

$$\mathbf{T} = (\mathrm{circ}(\mathbf{B}))^{-1}\mathbf{A}\,. \tag{4.4}$$

It can be seen from (4.4) that $\mathbf{T}$ can only be calculated, if $\mathrm{circ}(\mathbf{B})$ is a non-singular matrix. This means for arbitrary vectors $\mathbf{A}$ and $\mathbf{B}$, the exact transformation vector $\mathbf{T}$ might not always exist. However, it can be shown that a circulant matrix $\mathrm{circ}(\mathbf{B})$ is non-singular with probability 1 as long as the distribution of elements of the vector $\mathbf{B}$ is defined by a density function[1] which is the case for HRR vectors.

4.2 Transformation of a Class of Structures

Using circulant matrices allows us to calculate the exact transformation vector that transforms a single structure into another one. For example, we can find a transforma-

[1]The argument is based on the observation that the probability of $\mathbf{F_k}\mathbf{B} = 0$ is zero for all $k = 1 \ldots n$, where $\mathbf{F_k}$ is the k-th column of the Fourier matrix and $\mathbf{F_k}\mathbf{B}$ are the eigenvalues of $\mathrm{circ}(\mathbf{B})$. Under the condition that neither $\mathbf{F_k}$ nor $\mathbf{B}$ is the null vector, the equation $\mathbf{F_k}\mathbf{B} = 0$ describes a $n-1$-dimensional hyperplane in a n-dimensional space. A $n-1$-dimensional hyperplane, however, has n-dimensional Lebesgue measure 0 (Billingley, 1995), i.e., its probability in the n-dimensional space is 0.

tion vector that transforms the representation of the structure $p \rightarrow q$ into the logically equivalent structure $\neg p \vee q$. In Chapter 3 we discussed a method of constructing a transformation vector which can be applied to the transformation of a whole class of structures. We can, for example, construct a vector that transforms structures of the class $x_1 \rightarrow x_2$ into structures of the class $\neg x_1 \vee x_2$ with $x_1, x_2 \in \{p, q, r, s \ldots\}$. Since the exact transformation vector is unique for each corresponding pair of structures, we cannot construct a transformation vector that transforms all structures of a class perfectly. However, the decoding of HRRs can handle a certain amount of noise in the representations. A perfect transformation of all structures is therefore not necessary. The transformation vector only needs to be able to transform the structures such that the results are still recognisable. Clearly, the best transformation vector for a class of structures is the vector that minimises the amount of noise introduced to the result structures, i.e., the error of the transformation. How can such a transformation vector be learned from examples? We first need to define a suitable measure for the error of the transformation. This error can then be minimised with respect to the transformation vector for the particular example set used. A commonly used method for minimising an error function is gradient descent.

4.2.1 Learning by Gradient Descent

Learning in a system can be formulated in terms of the minimisation of an error E, where E is a function of a vector of adaptive parameters $\mathbf{X} = (x_1, \ldots x_n)$. A typical error function is the *Mean Square Error* that calculates the sum of the errors from all individual training examples for the learning system. The parameter vector that minimises $E(\mathbf{X})$ for a given set of inputs and outputs of the system is the solution for the learning problem. Under the condition that the error function is continuous and differentiable, *Gradient Descent* can be used to find a parameter vector that locally minimises the error function. Starting with a randomly chosen parameter vector, the gradient $\nabla\mathbf{E}$ of the error function is calculated. The gradient is the vector of the derivatives of E with respect to the components of $\mathbf{X}$:

$$\nabla\mathbf{E} = \left(\frac{\partial E}{\partial x_1}, \frac{\partial E}{\partial x_2}, \ldots, \frac{\partial E}{\partial x_n}\right). \tag{4.5}$$

The gradient of the error function specifies the direction of the steepest increase in E (Mitchell, 1997). Therefore, in order to minimise the error the parameter vector is altered by the negative of the gradient:

$$\mathbf{X}_{new} = \mathbf{X}_{old} - \eta \nabla \mathbf{E} . \tag{4.6}$$

The *learning rate* η is a small positive constant specifying the step size of the descent. After updating the vector $\mathbf{X}$ the process is repeated. Under the condition that the error function has only a global minimum, the algorithm converges if the learning rate η is small enough.

We can now apply gradient descent to learning the transformation of a class of structures. For m given pairs of n-dimensional structure vectors $\{\mathbf{A}_i, \mathbf{B}_i\}$ we want to find the transformation vector $\mathbf{T} = (t_1, \ldots, t_n)$ such that $\mathbf{A}_i = \mathbf{B}_i \circledast \mathbf{T}$ for all $i = 1 \ldots m$. The mean square error of the transformation can be defined as

$$E = 1/2 \sum_{i=1}^{m} |\mathbf{A}_i - \mathbf{B}_i \circledast \mathbf{T}|^2 . \tag{4.7}$$

$\mathbf{A}_i$ is the target of a transformation and $\mathbf{B}_i \circledast \mathbf{T}$ its result when the transformation vector $\mathbf{T}$ is used. This error is a function of $\mathbf{T}$ with the gradient $\nabla \mathbf{E}$

$$\nabla \mathbf{E} = (\frac{\partial E}{\partial t_1}, \frac{\partial E}{\partial t_2}, \ldots, \frac{\partial E}{\partial t_n}) \quad \text{where} \tag{4.8}$$

$$\frac{\partial E}{\partial t_j} = \sum_{i=1}^{m} (\mathbf{B}_i \oplus (\mathbf{A}_i - \mathbf{B}_i \circledast \mathbf{T}))^{(j)} .$$

Here $^{(j)}$ denotes the jth component of the calculated vector. Thus, learning the transformation vector $\mathbf{T}$ can be performed as follows. Starting with a randomly chosen initial vector $\mathbf{T}$, iterate through all training examples and calculate the gradient of the error function. After each such iteration, update the transformation vector such that

$$\mathbf{T}_{new} = \mathbf{T}_{old} + \Delta \mathbf{T} \quad \text{with} \tag{4.9}$$

$$\Delta \mathbf{T} = -\eta \sum_{i=1}^{m} (\mathbf{B}_i \oplus (\mathbf{A}_i - \mathbf{B}_i \circledast \mathbf{T})) . \tag{4.10}$$

The algorithm is only guaranteed to converge to the global minimum, if there do not exist any local minima on the surface of the error function. We will defer the

question of local minima in our error function to the next Section, where we develop a single-shot learning algorithm for the transformation of a class of structures.

As argued before, any transformation vector $\mathbf{T}$ can only transform a single structure perfectly. The mean square error for learning the transformation of a class of structures will therefore not converge to 0. This means that we need a suitable condition to stop the learning process. In the simplest case, learning could be continued for a predefined number of iterations. However, this criterion does not tell us anything about the quality of the learned transformation vector. As for backpropagation learning in feedforward networks, other sensible convergence criteria can be derived from the following observations: At the minimum of the error function, the gradient of the error function is 0 and the error function is stationary. Suitable stopping criteria can therefore be formulated as follows (cf. Haykin (1999)):

1. The gradient of the error function reaches a sufficiently small threshold, i.e., $\nabla \mathbf{E} < \epsilon_1$, where ϵ_1 is a predefined small value.

2. The rate of change in E in each training cycle reaches a sufficiently small threshold, i.e., $\Delta E < \epsilon_2$, where ϵ_2 is a predefined small value.

It should be noted, however, that these criteria can lead to unnecessarily long training times or premature termination of the learning process, if ϵ_1 and ϵ_2 are chosen too small or too large. A third possibility is to test the learned transformation vector after each iteration and to continue learning until all training examples are correctly transformed.

Example 1

For the tests of our learning algorithms we chose again the domain of propositional logic. We first applied the learning method to the transformation of a single structure. This allows us to compare the learned transformation vector with the exact transformation vector calculated using Equation (4.4). A transformation vector was learned that transforms the structure $p \rightarrow q$ into the logically equivalent structure $\neg p \vee q$. Structure vectors were formed as follows. The vectors { **p, q, op, not, disj, first, second, impl, ante, cons** } were randomly chosen 4096-dimensional HRR vectors representing the roles and fillers of the expressions. Then

$$\mathbf{not\_p} = (\mathbf{not} \circledast \mathbf{p})$$

$$\mathbf{p\_impl\_q} = (\mathbf{op} \circledast \mathbf{impl} + \mathbf{ante} \circledast \mathbf{p} + \mathbf{cons} \circledast \mathbf{q})$$

$$\mathbf{not\_p\_or\_q} = (\mathbf{op} \circledast \mathbf{disj} + \mathbf{first} \circledast \mathbf{not\_p} + \mathbf{second} \circledast \mathbf{q}).$$

Figure 4.1 shows the mean square error when training was performed for 500 iterations with a learning rate $\eta = 0.01$. As can be seen, the error converged very slowly.

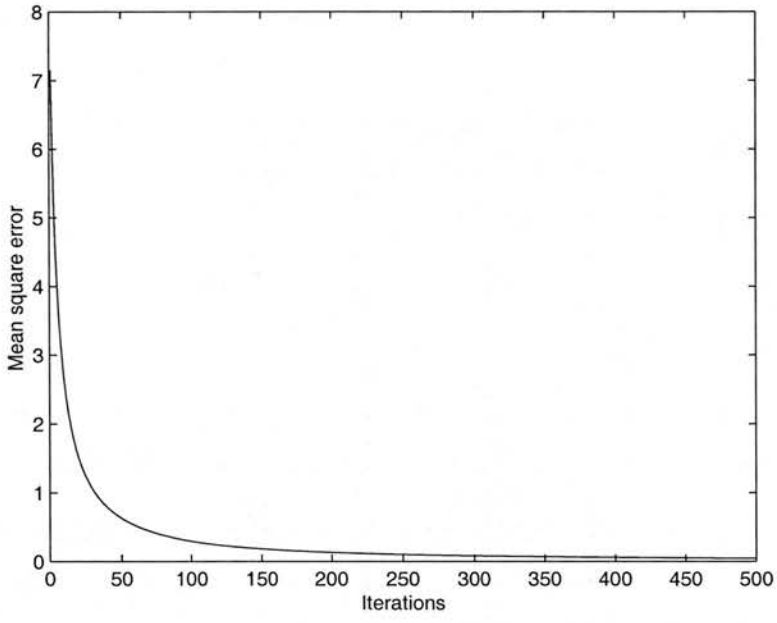

Figure 4.1: Mean square error when learning the transformation $p \rightarrow q \Longrightarrow \neg p \vee q$ using gradient descent. Training was performed for 500 iterations with a learning rate $\eta = 0.01$.

After 500 iterations, the mean square error was still as large as 0.046. The Euclidean distance between the learned and the calculated exact transformation vector was 1.52, the normalised dot product was 0.78.

Table 4.1 shows the mean square error and the similarity of the learned and the calculated transformation vector when training was continued for up to 50000 iterations. As learning progressed the learned transformation vector became more and more similar to the calculated exact transformation vector and the mean square error converged slowly to 0.

We can further test how accurately the learned vector transforms the input structure

Iterations	Dot product	Euclidean distance	Mean square error
500	0.78	1.52	0.046
2000	0.88	1.10	0.0077
5000	0.93	0.85	0.0023
10000	0.95	0.67	$8.93 * 10^{-4}$
50000	0.99	0.04	$1.52 * 10^{-6}$

Table 4.1: Learning the transformation $p \rightarrow q \Longrightarrow \neg p \vee q$ by gradient descent. The dot product and Euclidean distance between the learned and the calculated exact transformation vector and the mean square error for learning are shown.

depending on the number of iterations used for training. Table 4.2 shows the similarity of the target vector and the result vector when the learned transformation vector was applied to the input structure. After 500 iterations of gradient descent the Euclidean distance between the result of the transformation and the target was still as high as 0.39. This similarity was sufficient to decode the result correctly in this small test domain, i.e., we could retrieve all elements of the transformed structure successfully. However, the noise introduced to the transformed structure was relatively high, which could be problematic if in a larger system such a transformed structure is going to be used for further manipulation, possibly involving more transformations. Only after a much larger number of iterations does the transformation become more accurate. For comparison, when the exact transformation vector calculated from Equation (4.4) is used, the dot product between the result and the target is 1.0, the Euclidean distance is $2.41 * 10^{-12}$.

Iterations	Dot product	Euclidean distance
500	0.955	0.39
2000	0.975	0.1237
5000	0.9992	0.0678
10000	0.9997	0.0423
50000	1.0	0.0017

Table 4.2: Similarity of the target vector and the result vector of the transformation $p \rightarrow q \Longrightarrow \neg p \vee q$ learned by gradient descent for an increasing number of iterations.

Example 2

In a second simulation we applied the learning algorithm to the transformation of a class of structures. Gradient descent was used to learn the transformation $x_1 \rightarrow x_2 \Longrightarrow \neg x_1 \vee x_2$ with $x_1, x_2 \in \{p, q, r, s\}$. The input and output structures were formed as in Example 1 but for all possible combinations of the filler vectors { **p, q, r, s** }. Figure 4.2 shows the mean square error when training was performed for 500 iterations through all training examples with a learning rate $\eta = 0.002$. Since the exact transformation vector is unique for each structure pair, we expected the error for the transformation to be significantly larger than the error for the transformation of a single structure. Recall that the mean square error is calculated in each training cycle by the sum over the errors for each structure pair. After 500 iterations the mean square error was 12.81. However, the transformation vector learned after 500 iterations transformed all input structures correctly, i.e., the result structures of the transformation were correctly decoded into their elements.

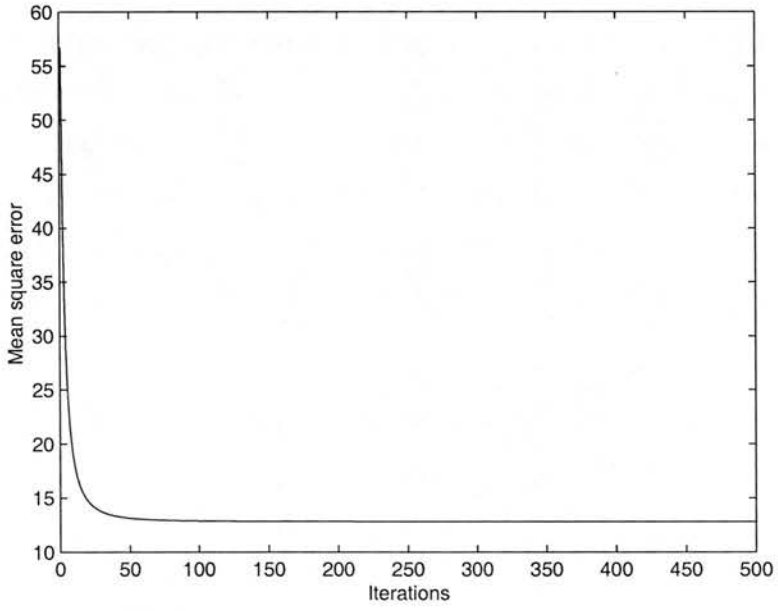

Figure 4.2: Mean square error when learning the transformation $x_1 \rightarrow x_2 \Longrightarrow \neg x_1 \vee x_2$ for $x_1, x_2 \in \{p, q, r, s\}$. Gradient descent learning was performed for 500 iterations with a learning rate $\eta = 0.002$.

4.2.2 One-shot Learning

The examples above show that gradient descent learning converges only very slowly at the solution for the learning problems. Looking for a faster solution we observed that the gradient of the mean square error for a transformation can also be used in a learning method where the transformation vector $\mathbf{T}$ can be acquired in a single pass through all training examples. For m given pairs of n-dimensional structure vectors $\{\mathbf{A}_i, \mathbf{B}_i\}$ we want to find the transformation vector $\mathbf{T} = (t_1, \ldots t_n)$ such that $\mathbf{A}_i = \mathbf{B}_i \circledast \mathbf{T}$ for all $i = 1 \ldots m$. We start with the observation that for an optimal transformation vector $\mathbf{T}$ the gradient of the error function should be 0:

$$\nabla \mathbf{E} = 0 . \tag{4.11}$$

Using the mean square error function as defined in Equation (4.7) and its gradient (4.8) we get

$$\sum_{i=1}^{m} (\mathbf{B}_i \oplus (\mathbf{A}_i - \mathbf{B}_i \circledast \mathbf{T})) = 0 \tag{4.12}$$

$$\sum_{i=1}^{m} (\mathbf{B}_i \oplus \mathbf{A}_i) - \sum_{i=1}^{m} (\mathbf{B}_i \oplus (\mathbf{B}_i \circledast \mathbf{T})) = 0 . \tag{4.13}$$

This leads to

$$\underbrace{\sum_{i=1}^{m} (\mathbf{B}_i \oplus \mathbf{A}_i)}_{\mathbf{V}} = \underbrace{\sum_{i=1}^{m} (\mathbf{B}_i \oplus \mathbf{B}_i)}_{\mathbf{U}} \circledast \mathbf{T} . \tag{4.14}$$

Then

$$\mathbf{V} = \mathbf{U} \circledast \mathbf{T} . \tag{4.15}$$

As Equation (4.1), describing the transformation of a single structure, this equation can be solved using a circulant matrix. The circular convolution of the two vectors $\mathbf{U}$ and $\mathbf{T}$ can be written as the multiplication of the matrix $\mathrm{circ}(\mathbf{U})$ and the vector $\mathbf{T}$:

$$\mathbf{V} = \mathrm{circ}(\mathbf{U})\, \mathbf{T} \tag{4.16}$$

$$\mathbf{T} = (\mathrm{circ}(\mathbf{U}))^{-1} \mathbf{V} . \tag{4.17}$$

Again, the distribution of the vector elements of $\mathbf{U}$ can be described by a density function. The matrix $\text{circ}(\mathbf{U})$ is therefore non-singular with probability 1. Equation (4.17) can be solved using Fast Fourier Transforms. The Fourier Matrix $\mathbf{F}$ diagonalises $\text{circ}(\mathbf{U})$ (Davis, 1979) such that

$$\text{circ}(\mathbf{U}) = \mathbf{F}^{-1}\text{diag}(\mathbf{FU})\mathbf{F}\,. \tag{4.18}$$

This leads to

$$\begin{aligned} \text{circ}(\mathbf{U})^{-1} &= (\mathbf{F}^{-1}\text{diag}(\mathbf{FU})\mathbf{F})^{-1} && (4.19)\\ &= \mathbf{F}^{-1}(\mathbf{F}^{-1}\text{diag}(\mathbf{FU}))^{-1} && (4.20)\\ &= \mathbf{F}^{-1}\text{diag}(\mathbf{FU})^{-1}\mathbf{F}\,. && (4.21) \end{aligned}$$

With (4.17) and (4.21) we get

$$\begin{aligned} \mathbf{T} &= \mathbf{F}^{-1}\text{diag}(\mathbf{FU})^{-1}\mathbf{FV} && (4.22)\\ &= \mathbf{F}^{-1}((\mathbf{FV})./(\mathbf{FU}))\,. && (4.23) \end{aligned}$$

Here $./$ denotes the element-wise division of two vectors.

We see now that in order to obtain $\mathbf{T}$ we only have to pass through all training examples once, calculating the sums $\mathbf{U}$ and $\mathbf{V}$. We then solve Equation (4.23) which only requires two Fast Fourier Transforms, one inverse Fast Fourier Transform, and one element-wise division of two vectors. Note that the Fast Fourier Transform of a n-dimensional vector requires only $\text{O}(n \log n)$ operations. Thus, this method provides a very efficient way of learning the transformation vector for a class of structures from examples.

Using Equation (4.23) we can also answer the question of local minima in our error function as defined in Equation (4.7). Gradient descent might not converge to the optimal solution but get stuck in one such minimum. However, this is not the case for our error function. A characteristic of a minimum, be it local or global, is that $\nabla\mathbf{E} = 0$. If our error function had local minima, then for a given set of vectors $\{\mathbf{A}_i, \mathbf{B}_i\}$ more than one transformation vector $\mathbf{T}$ would satisfy this condition, i.e., more than one vector $\mathbf{T}$ would be a solution of Equation (4.12). However, $\mathbf{T}$ is a unique solution of Equation (4.23) which was derived from Equation (4.12), i.e., from exactly the assumption $\nabla\mathbf{E} = 0$. We can therefore conclude that the mean square

error function for our learning problem as defined in Equation (4.7) has only a global minimum.

Note that the one-shot learning described above reduces to calculating the transformation vector using Equation (4.4) when applied to a single pair of vectors. In this case, $m = 1$ and

$$\nabla \mathbf{E} = 0 \tag{4.24}$$

$$(\mathbf{B} \oplus (\mathbf{A} - \mathbf{B} \circledast \mathbf{T})) = 0 \tag{4.25}$$

$$(\mathbf{B} \oplus \mathbf{A}) - (\mathbf{B} \oplus (\mathbf{B} \circledast \mathbf{T})) = 0\,. \tag{4.26}$$

Now, Equation (4.14) reduces to

$$\mathbf{B} \oplus \mathbf{A} = \mathbf{B} \oplus \mathbf{B} \circledast \mathbf{T} \tag{4.27}$$

$$\mathbf{A} = \mathbf{B} \circledast \mathbf{T} \tag{4.28}$$

$$\mathbf{T} = (\mathrm{circ}(\mathbf{B}))^{-1}\mathbf{A}\,. \tag{4.29}$$

Example 3

We again used the transformation $x_1 \rightarrow x_2 \Longrightarrow \neg x_1 \vee x_2$ with $x_1, x_2 \in \{p, q, r, s\}$ from Example 2, now applying the one-shot learning method. The learned transformation vector correctly transformed the input structures, i.e., after the transformation all elements of all structures were correctly decoded. We then compared the transformation vector with the vector learned in Example 2 using gradient descent. Table 4.3 shows the similarity of the transformation vectors acquired by both methods. We also show the mean square error for gradient descent. Gradient descent learning was performed for up to 10000 iterations. It can be seen that although the error does seem to be stationary, the similarity between the gradient descent vector and the one-shot vector increases with the number of iterations in gradient descent learning[2]. For a sufficiently large number of iterations of gradient descent, both methods learn the same transformation vector if the same training set is used.

[2]This example shows that the condition on the termination of the learning process is crucial for the successful acquisition of the transformation vector by gradient descent. Using the rate of change in the error in this example would halt the training prematurely.

Iterations	Dot product	Euclidean distance	Mean square error
500	0.994	0.09	12.81
2000	0.999	0.01	12.80
5000	1.000	$1.9 * 10^{-4}$	12.80
10000	1.000	$3.2165 * 10^{-7}$	12.80

Table 4.3: Similarity of the transformation vectors learned with gradient descent and the one-shot learning method for the transformation $x_1 \rightarrow x_2 \Longrightarrow \neg x_1 \vee x_2$ with $x_1, x_2 \in \{p, q, r, s\}$ and an increasing number of iterations for gradient descent.

4.3 Systematicity and Generalisation

In both classical artificial intelligence and Connectionism, the quality of a learning system is usually measured by its capacity to generalise the acquired knowledge to previously unseen examples. We have seen in Chapter 3 that connectionist architectures are, in principle, capable of generalisations that require processes of level 3 systematicity. Niklasson (1993) presented a network that learned the holistic transformation of structure representations formed in the hidden layer of a RAAM and generalised the acquired knowledge to structures containing novel elements. However, the generalisation was based on the similarity of representations for elements, i.e., the initial representations for elements were carefully constructed by hand such that elements filling the same roles in the structures had similar representations. Further, the encoding of RAAM structures and their transformation were learned in parallel. This means that the representations of structures formed in the hidden layer of the RAAM were optimised with respect to the transformation task. Although RAAM representations formed the basis of the transformation process, their encoding in the RAAM alone was not sufficient to support generalisations that require level 3 systematicity.

We used Niklasson's transformation task to test the generalisation capacity of our new methods learning the transformation of HRRs. Our simulations differed from Niklasson's experiment in two ways: Randomly chosen HRR vectors represented the elements in the domain, and the formation of structures and their transformation were performed independently. 512 structure pairs were composed for the transformations

$$(x_1 \; op \; x_2) \rightarrow x_3 \iff \neg (x_1 \; op \; x_2) \vee x_3$$

$$x_1 \rightarrow (x_2 \; op \; x_3) \iff \neg x_1 \vee (x_2 \; op \; x_3)$$

with $x_1, x_2, x_3 \in \{p, q, r, s\}$ and $op \in \{\rightarrow, \vee\}$. The training set consisted of all 216 structure pairs not containing the element s, the remaining pairs formed the test set. The transformation vector was learned using the one-shot learning method. The correctness of each transformation was tested by decomposing the transformed structure into its components and comparing the results with the element vectors in the domain. After training, the transformation vector correctly transformed all examples in the training and test set. This result shows that our method can generalise to structures containing unseen elements, a task which requires a systematicity of level 3 and could only be learned for RAAM structures using an interleaved training of the RAAM and the transformation network. Note that this result was also achieved performing gradient descent learning for 100 iterations through all training examples. A comparison of the results showed that both methods converged at the same transformation vector. After 100 iterations of gradient descent, the Euclidean distance between the transformation vectors learned with both methods was $8.96{*}10^{-5}$ and the normalised dot product was 0.99.

Our second experiment was designed to test the generalisation of the learned transformation to structures of a higher complexity. Successfully solving this task requires systematicity of level 4 for structures not containing novel elements and of level 5 for structures including novel elements. A transformation vector was first learned for the simple transformation

$$x_1 \rightarrow x_2 \iff \neg x_1 \vee x_2$$

with $x_1, x_2 \in \{p, q, r\}$. Again, randomly generated element vectors were used and the resultant structures were decomposed into their elements. The transformation vector transformed all 18 training examples correctly. As in the previous experiment, the transformation vector also generalised with 100% accuracy when applied to test examples of the same complexity but containing the element s not seen during training. 10 test sets containing structures of higher complexity than the training examples were then constructed with $x_1, x_2, x_3, x_4 \in \{p, q, r, s\}$. The test sets are shown in Table 4.4 together with the generalisation error, i.e., the relative number of wrongly transformed elements in all structures of the test set. The transformed structures of the first six

Tested Structure	Error in %
$(x_1 \wedge x_2) \rightarrow x_3 \Longleftrightarrow \neg (x_1 \wedge x_2) \vee x_3$	0
$(x_1 \vee x_2) \rightarrow x_3 \Longleftrightarrow \neg (x_1 \vee x_2) \vee x_3$	0
$(x_1 \rightarrow x_2) \rightarrow x_3 \Longleftrightarrow \neg (x_1 \rightarrow x_2) \vee x_3$	0
$x_1 \rightarrow (x_2 \wedge x_3) \Longleftrightarrow \neg x_1 \vee (x_2 \wedge x_3)$	0
$x_1 \rightarrow (x_2 \vee x_3) \Longleftrightarrow \neg x_1 \vee (x_2 \vee x_3)$	0.78
$x_1 \rightarrow (x_2 \rightarrow x_3) \Longleftrightarrow \neg x_1 \vee (x_2 \rightarrow x_3)$	0.16
$(x_1 \wedge x_2) \rightarrow (x_3 \wedge x_4) \Longleftrightarrow \neg (x_1 \wedge x_2) \vee (x_3 \wedge x_4)$	0.11
$(x_1 \wedge x_2) \rightarrow (x_3 \rightarrow x_4) \Longleftrightarrow \neg (x_1 \wedge x_2) \vee (x_3 \rightarrow x_4)$	0.28
$((x_1 \wedge x_2) \rightarrow x_3) \rightarrow x_4 \Longleftrightarrow \neg ((x_1 \wedge x_2) \rightarrow x_3) \vee x_4$	0.47
$((x_1 \wedge x_2) \wedge x_3) \rightarrow x_4 \Longleftrightarrow \neg ((x_1 \wedge x_2) \wedge x_3) \vee x_4$	4.35

Table 4.4: Generalisation to examples of higher complexity. The transformation vector was learned for the transformation $x_1 \rightarrow x_2 \Longleftrightarrow \neg x_1 \vee x_2$ with $x_1, x_2 \in \{p, q, r\}$ using the one-shot learning method.

test sets contained 640 elements in total[3]. The remaining test sets each contained 3584 elements in the transformed structures. Note that the connective $\wedge$ was not part of the training set. As Table 4.4 shows, the acquired knowledge was generalised to unseen examples more complex than the training set and containing novel elements with very high accuracy. This task requires a systematicity of level 5.

For comparison we repeated the experiment using a constructed transformation vector. The HRR vectors representing the structures were constructed from the role and filler vectors as shown in Example 1, and the transformation vector $\mathbf{T}$ was constructed as follows:

$$\mathbf{T}_1 = (\,\mathbf{impl}' \circledast \mathbf{disj}\,) + (\,\mathbf{ante}' \circledast \mathbf{not} \circledast \mathbf{first}\,) + (\,\mathbf{cons}' \circledast \mathbf{second}\,)$$

$$\mathbf{T}_2 = (\,\mathbf{disj}' \circledast \mathbf{impl}\,) + (\,\mathbf{first}' \circledast \mathbf{not}' \circledast \mathbf{ante}\,) + (\,\mathbf{second}' \circledast \mathbf{cons}\,)$$

$$\mathbf{T} = \mathbf{T}_1 + \mathbf{T}_2$$

Note that this is the same transformation vector as in our experiments in Section 3.3.2.

[3]This is calculated from 4^3 different variations of the values for x_1, x_2, x_3 in a class of result structures $\times$ 5 decoded elements per variation $\times$ 2 classes of result structures per transformation.

This vector performed the transformation $x_1 \rightarrow x_2 \iff \neg x_1 \vee x_2$ with 100% accuracy when 4096-dimensional vectors were used. Table 4.5 shows the generalisation performance of the constructed transformation vector to the test structures of higher complexity. As can be seen, the errors are of the same magnitude as the errors pro-

Tested Structure	Error in %
$(x_1 \wedge x_2) \rightarrow x_3 \iff \neg (x_1 \wedge x_2) \vee x_3$	0
$(x_1 \vee x_2) \rightarrow x_3 \iff \neg (x_1 \vee x_2) \vee x_3$	1.09
$(x_1 \rightarrow x_2) \rightarrow x_3 \iff \neg (x_1 \rightarrow x_2) \vee x_3$	0
$x_1 \rightarrow (x_2 \wedge x_3) \iff \neg x_1 \vee (x_2 \wedge x_3)$	0
$x_1 \rightarrow (x_2 \vee x_3) \iff \neg x_1 \vee (x_2 \vee x_3)$	0
$x_1 \rightarrow (x_2 \rightarrow x_3) \iff \neg x_1 \vee (x_2 \rightarrow x_3)$	0
$(x_1 \wedge x_2) \rightarrow (x_3 \wedge x_4) \iff \neg (x_1 \wedge x_2) \vee (x_3 \wedge x_4)$	0.86
$(x_1 \wedge x_2) \rightarrow (x_3 \rightarrow x_4) \iff \neg (x_1 \wedge x_2) \vee (x_3 \rightarrow x_4)$	0.05
$((x_1 \wedge x_2) \rightarrow x_3) \rightarrow x_4 \iff \neg ((x_1 \wedge x_2) \rightarrow x_3) \vee x_4$	2.40
$((x_1 \wedge x_2) \wedge x_3) \rightarrow x_4 \iff \neg ((x_1 \wedge x_2) \wedge x_3) \vee x_4$	0.17

Table 4.5: Generalisation to examples of higher complexity. The transformation vector was constructed for the transformation $x_1 \rightarrow x_2 \iff \neg x_1 \vee x_2$.

duced with the learned transformation vector. The constructed and the learned transformation vector were not identical, however. The normalised dot product between the learned and the constructed transformation vector was 0.75 for the original three variables p, q, and r in the training set. When more training examples including more variables in the structures were used for one-shot learning, the learned transformation vector became more similar to the constructed one. The dot product of the two vectors increased to 0.81 for 10 variables and 0.83 for 15 and 20 variables in the training set. For this transformation, the constructed transformation vector seems to be a fairly good approximation of the optimal one.

For both the learned and the constructed transformation vector, most decoding errors were observed for elements in the lowest level of embedding, such as x_1 in the expression $\neg ((x_1 \wedge x_2) \wedge x_3) \vee x_4$. Interestingly, decoding errors were primarily found for the variables $x_1 \ldots x_4$ and the novel connective $\wedge$ when the constructed transfor-

mation vector was used, but mainly for connectives when the learned transformation vector was used. In both cases, an erroneous element was usually decoded as another element of the same type, i.e., either as a different variable or a wrong connective.

4.4 Discussion

We presented two different methods for learning the holistic transformation of hierarchical structures represented by HRRs from examples. Both methods have advantages and disadvantages. Gradient descent is a simple and well understood method that can be straightforwardly implemented in a neural architecture. In fact, Plate (1994) designed a recurrent neural network that learns to store sequences of elements by associating the elements with points along a predefined trajectory. Both the points along the trajectory and the elements were represented by HRR vectors and the network was trained using gradient descent.

The key problem of learning by gradient descent is the termination of the learning process. We have seen that for the transformation of a single structure the error converges to 0. However, this usually requires a large number of training cycles. For the transformation of a class of structures, the error is expected to be much larger than 0 and suitable criteria for stopping the training process have to be found. Which criterion is most suitable depends on the actual task to be learned. Whereas in Example 1 the rate of change in the error might be a good criterion for stopping the learning process, we have seen in Example 2 and 3 (cf. Figure 4.2 and Table 4.3) that despite the fact that the rate of change in E is very close to 0, the learned transformation vector still improves significantly if training is continued. Using the rate of change in E here might terminate the learning process prematurely.

The one-shot learning algorithm, on the other hand, is fast and efficient. It requires only one iteration through all training examples and arrives at the optimal solution for the training set. It is not obvious to us, however, how a learning method involving Fast Fourier Transforms could be easily implemented into a standard connectionist architecture.

Note that the results obtained in our simulations were very consistent when the simulations were repeated several times with different randomly generated element

vectors. Decoding errors for elements occurred consistently in the same positions and were of the same magnitude as the errors in the examples presented. The same was observed for all experiments reported in the following Chapters.

Our system is capable of generalising the holistic transformation of HRR structures to structures of higher complexity than the training examples and, like Niklasson's (1993) and Niklasson and van Gelder's (1994a) RAAM and backpropagation network, to structures containing novel elements. An interesting question regarding the latter case is what such generalisation to novel elements is based on. As noted in Chapter 3, Niklasson and van Gelder (1994b) stress the importance of the initial representation for elements in a domain, pointing out that the generalisation to novel elements in a structure can only be based on the similarity of element representation. With an application of their architecture to natural language processing in mind they state (Niklasson and van Gelder, 1994b):

> "It would be impossible for a network to correctly process a sentence containing a novel constituent if it had no information at all concerning the syntactic category of that constituent. (This would be like somebody asking you to use the word 'pilk' correctly in English sentences without even telling you whether it is a noun or a verb.)"

They conclude:

> "One way of handling this problem is to carefully choose, by hand, basic representations for the atomic constituents which reflect their syntactic category."

Therefore, Niklasson and van Gelder use element representations which are either completely constructed by hand (Niklasson, 1993) or formed in a separate RAAM in a way that strongly reflects the similarity of elements (Niklasson and van Gelder, 1994b). In both cases, representations of novel elements are very similar to representations of elements that fall into the same syntactic category in the training examples. We agree that some information of the syntactic category of novel elements is necessary in order to correctly process structures containing them. However, we do not believe that this information has to be provided explicitly. As Hadley (1994) points out, once they have acquired language

> "...humans do in fact induce the syntactic role of many words without being told, explicitly, what the role is."

To use Niklasson and van Gelder's analogy, humans are capable of, for example, correctly transforming the sentences 'John left the pilk on the bus.' and 'We pilk the washing machine.' into their corresponding passive constructions without having heard of 'pilk' before. It seems to us that humans acquire the knowledge for this transformation not so much from the similarity of the representations of words but from the syntactic role these words play in the sentence, which is inferred from the relations between the words in the sentence. The same seems to be the case in our system. Since it uses randomly generated representations for elements, its generalisation capacity cannot be based on the similarity of these representations. Rather, the knowledge necessary for such generalisation is inferred from the relation of elements within a structure. The syntactic category of an element does not need to be provided explicitly.

Using randomly generated representations for elements carries another advantage. Hadley (1994) points out that Niklasson (1993) demonstrated the generalisation performance of his system using a nearly exhaustive training set. Only 1 out of 4 variables was used as the novel element. He concludes (Hadley, 1994, original emphasis):

> "A question we will want to consider, then, is whether the existence of c-nets [connectionist networks] that successfully generalise on the basis of nearly exhaustive training sets entails the *in-principle* possibility of c-nets that generalise on the basis of much sparser training sets."

Whereas extending a domain when using hand-crafted element representations is difficult, using randomly generated representations for elements means that a domain can easily be extended by another hundred or thousand randomly generated elements. Such a large scale experiment has not yet been carried out. However, given the good scaling properties of HRR in general, we would expect that the result will not differ from the results of our experiments with relatively small test sets.

We observed in Chapter 3 that the transformation of HRR structures by a constructed transformation vector is problematic if parts of the input structures are supposed to remain unchanged. Recall that we were not able to construct a transformation

vector that changes structures of the form

$$\mathbf{S}_i = \mathbf{r}_1 \circledast \mathbf{X} + \mathbf{r}_2 \circledast \mathbf{Y}_i$$

into structures of the form

$$\mathbf{S}_i = \mathbf{r}_1 \circledast \mathbf{A} + \mathbf{r}_2 \circledast \mathbf{Y}_i .$$

Out of 46 structures only 9 were correctly transformed. We repeated the experiment learning the transformation vector from two example structures and testing it on the remaining 44. All structures were transformed and decoded correctly.

It is somewhat difficult to analyse a learned transformation vector with respect to the components it contains, because the extraction of a particular element from a structure requires knowledge about possible elements it is bound to in the structure. However, we are at least able to say that the learned transformation vector contained, like the constructed one, the term $\mathbf{X}' \circledast \mathbf{A}$, because we decoded $\mathbf{T} \circledast \mathbf{X} \approx \mathbf{A}$ and $\mathbf{T} \circledast \mathbf{A}' \approx \mathbf{X}'$, which means that $\mathbf{X}'$ is bound to $\mathbf{A}$ in the transformation vector $\mathbf{T}$. However, the learned transformation vector was not identical with the constructed transformation vector from Chapter 3. We would expect that when the system learns the transformation vector, it is able to pick up subtleties in the input and target structures and incorporates them into the learned vector that a constructed transformation vector does not capture. This knowledge enabled the learned transformation vector to transform parts of the structures while leaving others unchanged.

The experiments presented here are only a first step into the investigation of holistic transformations of HRR structures. We believe that transformations of this kind could provide a means for efficiently solving more complex problems that require a high degree of systematicity, such as logical inference. However, performing a chain of inference clearly involves more than a single structural transformation of logical formulae. A number of *different* transformation rules has to be acquired and appropriately applied by the system. Inference also requires not only the transformation of logical formulae, but their combination and reduction. Finally, an inference system has to provide mechanisms for distinguishing valid from invalid inferences. For example, in order to correctly apply Modus Ponens given the formulae $x_1 \rightarrow x_2$ and x_1, the system has to combine the two formulae and reduce the combination $x_1 \rightarrow x_2, x_1$

to x_2, while at the same time rejecting the application of the rule to the formulae $x_1 \rightarrow x_2$ and x_3, if x_1 and x_3 are not equivalent. We have investigated the first step towards such an inference system, namely the acquisition of a number of different transformational inference rules. The results of this work are presented in Chapter 6.

4.5 Summary

We developed two methods that holistically transform hierarchical structures represented by HRRs. In contrast to the construction of a transformation vector, our methods do not rely on any explicitly provided knowledge about the way the hierarchical structures are formed and about the modifications needed for their transformation. Further, the composition of structures and their transformation are performed separately, i.e., transformations are applied to hierarchical structures that are formed independently of the task they are used for.

We demonstrated that gradient descent can be used to learn the transformation from examples. Thus, a connectionist architecture is capable of learning and performing such transformation tasks. The gradient of the error function for the transformation can also be used to find the transformation vector more efficiently, iterating through all training examples only once.

Our experiments suggest that HRRs are more suitable for at least some high-level holistic processing than RAAM structures. In the direct comparison of a holistic transformation of HRR and RAAM representations the transformation could directly be applied to the HRR representations without their further optimisation whereas the transformation of RAAM structures and their encoding had to be learned in parallel in order to achieve level 3 systematicity. Representations formed in the hidden layer of a RAAM have the strong advantage, however, that they can be learned from representations for elements in a domain, while HRRs are constructed.

The results of our experiments contradict Fodor and Pylyshyn's (1988) claim that connectionist architectures are not capable of performing tasks which require a high degree of systematicity. Our system generalised acquired knowledge about the transformation of structures to structures containing novel elements and to structures of higher complexity than seen in the training set. This corresponds to level 5 system-

aticity as defined in Niklasson (1993), which, to the best of our knowledge, has not yet been achieved by any other comparable connectionist learning method.

Chapter 5

The Spatial Structure of HRRs and RAAM Representations

We have shown in Chapter 3 and 4 that connectionist representations can form the basis of systematic structure-sensitive processing. An open question still is, however, how the structural information about the objects that enables such systematic processing is captured in the flat, seemingly unstructured connectionist representations. The profoundly different nature of symbolic and connectionist representations suggests very different mechanisms underlying structure-sensitive processing within the two representational frameworks. In this Chapter we show that HRRs and representations learned by RAAMs structure the representational space such that clusters are formed of representations for structurally similar objects. We argue that it is this clustering that provides the information necessary for the systematic structure-sensitive processing of connectionist representations. Our argument is supported by the results obtained from clustering the representations for hierarchical structures underlying the experiments on holistic classification, matching, and transformation discussed in Chapter 3 and 4.

5.1 Constituent and Spatial Structure of Representations

The main difference between symbolic and connectionist representations of hierarchical structures is the way structures are composed from elements in a domain. Symbolic

representational schemes traditionally implement a concatenative mode of composition. This form of compositionality provides a straightforward explanation of the capacity to process the resulting representations in a systematic structure-sensitive manner. What is characteristic for concatenation is that the representation of a structured object contains the full representations of all constituents the object is comprised of or as van Gelder (1990, p. 360) puts it: "For a mode of combination to be concatenative, it must preserve tokens of an expression's constituents (and the sequential relations among tokens) in the expression itself." As we have argued in Chapter 2, what follows from the use of concatenation in symbolic systems is that the resulting representations exhibit an internal constituent structure. Structure-sensitive processes over symbolic representations for hierarchical objects then simply implement the manipulation of this constituent structure.

The connectionist representational schemes for structured objects investigated in the previous Chapters, on the other hand, are based on a merely functional compositionality. Here, the representations of composed objects do not contain the full representations of their constituents and do not themselves possess a constituent structure. However, van Gelder (1990, p. 361) rightly points out: "The absence of strictly syntactic [constituent] structure ... does not imply the absence of significant structure of any kind." The holistic operations explored in the previous Chapters clearly demonstrate that connectionist representations of hierarchical objects can be processed in a highly systematic structure-sensitive manner and thus give evidence for some kind of structure present in the underlying representations.

For representations formed in the hidden layer of RAAMs, this structure has been investigated by a number of researchers. Already when developing RAAMs, Pollack (1988) analysed the activations of the hidden layer units with respect to the features present in the hierarchical objects represented. He found that for a simple encoding task and with enough representational resources available, a RAAM can develop representations that show constituent structure. In this case, the information about the constituents of a structure is encoded by means of local representation. For example, after a 4-3-4 RAAM was trained to encode all possible sequences of three bits, each hidden unit in the RAAM encoded exactly one bit of the sequences. However, a RAAM has to employ a different strategy for representing structural information when there

are more constituents and their respective positions in a structure to be encoded than there are hidden units available in the network. To show this, Pollack (1988) trained a 20-10-20 RAAM to encode well-formed syntactic trees with five different elements as leaves. Now the number of combinations of values and positions of leaves exceeded the number of units and prohibited the local encoding of the complete structure of a tree. However, the analysis of the hidden unit activations revealed that some structural information was again encoded locally. The activation of one unit, for example, distinguished trees representing sentences from trees representing non-sentences. This local encoding of structural features was only partial, however. The learned representations did not show a complete constituent structure.

The same was observed by Niklasson and Bodén (1997) when analysing the representations formed by a 20-10-20 dual RAAM trained to represent binary trees (cf. Section 3.2.1). The activation of one hidden unit distinguished trees of depth 2 from trees of depth 3. The activation of another unit determined whether a certain element occurred as the right-most leaf in a left-balanced tree. However, other units did not show an obvious encoding of a single feature of the trees and some structural features could not be detected in the encoding at all. The localist encoding of only some structural features of the represented objects, however, cannot solely account for the structure-sensitive holistic operations they can be applied to.

Further analyses of the representations formed in the hidden layer of RAAMs were based on the observation that distributed representations learned by connectionist networks are similar to the extend that the objects they represent are similar. In general distributed representations learned by connectionist networks are real-valued vectors in a typically high-dimensional representational space. The similarity of such vectors can be understood in terms of their location in the representational space. Two vectors are similar if they are placed closely together in the representational space whereas dissimilar representations are well apart. Consequently, representations learned by connectionist networks should structure the representational space such that they form clusters of representations that stand for similar objects. This *spatial structure*, if it reflects the structural similarity of represented objects, however, could just like the constituent structure of symbolic representations form the basis for systematic structure-sensitive processing.

In order to test this hypothesis and to unveil and visualise the spatial structure of connectionist representations, different forms of clustering can be applied to the representational spaces.

5.2 Clustering

Clustering may be informally defined as the process of organising a number of objects into groups based on the characteristics the objects possess. The process should result in clusters such that the members of a cluster are very similar to each other according to some similarity measure, but are dissimilar to the members of other clusters. In our case, the characteristic according to which connectionist representations are to be clustered is their position in the representational space. Representations close in representational space should be members of the same cluster, whereas representations far apart from each other should be placed in different clusters.

5.2.1 Hierarchical and Non-Hierarchical Clustering

There is a vast number of different clustering algorithms available (see, e.g., Kaufman and Rousseeuw (1990) and Everitt (1993)). According to the approach used, they can usually be placed in either of the two groups *partitioning* and *hierarchical clustering.*

Partitioning methods, often also referred to as *k-clustering,* divide a set of m inputs $\mathcal{S} = \{s_1, s_2, \ldots, s_m\}$ into k subsets $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k$ such that each subset contains at least one object of $\mathcal{S}$ and each object $s_i \in \mathcal{S}$ belongs to exactly one subset. The k subsets thus cover the entire input space as shown in Figure 5.1a. The number of clusters has to be chosen before the actual clustering process, which requires the user of the method to have some idea about the clusters expected. This can prove to be problematic in some applications where only very little is known about the nature of the data. Alternatively, clustering can be performed for different values of k, and the clustering that appears to provide the most meaningful interpretation of the data is chosen afterwards (Kaufman and Rousseeuw, 1990). Further, the distance between clusters can serve as a guideline when choosing an appropriate number of clusters (Hair et al., 1995).

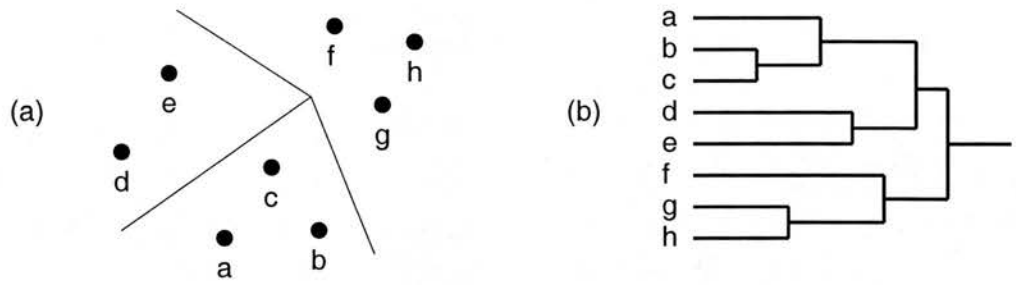

Figure 5.1: (a) Non-hierarchical partitioning with $m = 8$ and $k = 3$. (b) Dendrogram for a hierarchical clustering of 8 inputs. For agglomerative methods the dendrogram is read from left to right, for divisive methods from right to left.

Hierarchical clustering methods do not divide the input space in a single step. Rather, they construct a tree in which the nodes represent subsets of the input set. The top node of the tree represents the entire input set $\mathcal{S}$ while the leaves each represent a single member $s_i \in \mathcal{S}$ of the input. The tree can be constructed bottom-up or top-down. *Agglomerative* methods start with m clusters and in each subsequent step merge the two closest clusters into a new cluster, hereby reducing the number of clusters by one. In contrast, *divisive* methods start with the top node of the tree, i.e., the whole set $\mathcal{S}$ in one cluster, and in each following step the cluster containing the most dissimilar objects is split into two new clusters. The results of hierarchical clustering methods are usually presented in a dendrogram as shown in Figure 5.1b. Note that agglomerative and divisive methods can yield different results for the same data set.

The biggest disadvantage of hierarchical clustering methods is that sub-optimal decisions made in early stages of the process can not be corrected later. Once a agglomerative method has grouped two objects into the same cluster, they cannot be separated anymore, and once a divisive method has split a cluster, the objects in the two new clusters cannot be merged again.

Which clustering method to use strongly depends on the nature of the data to be clustered, the purpose of the clustering, expectations on the results, and preferences of the user. Various clustering methods, some implementing aspects of both hierarchical and non-hierarchical clustering, can also be found within the probabilistic and the connectionist framework.

5.2.2 Probability Density Estimation and EM

Within the probabilistic framework clustering can be viewed as identifying the dense regions of the probability density of the data source (Bradley et al., 1999). Since any distribution can be effectively approximated by a mixture of Gaussians (Silverman, 1985; Scott, 1992), the probability density function is most commonly represented by a *Gaussian Mixture Model* consisting of a number of d-dimensional Gaussians each representing a dense region or cluster. The Gaussian distribution for cluster i is parametrised by the mean vector $\boldsymbol{\mu}_i$ and the covariance matrix $\boldsymbol{\Sigma}_i$:

$$Pr(\mathbf{x}|i) = \frac{1}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma}_i|}} exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^T(\boldsymbol{\Sigma}_i)^{-1}(\mathbf{x}-\boldsymbol{\mu}_i)\right) \tag{5.1}$$

where $|\boldsymbol{\Sigma}_i|$ is the determinant of $\boldsymbol{\Sigma}_i$ and $(\boldsymbol{\Sigma}_i)^{-1}$ is its inverse. The mixture model probability density function is:

$$Pr(\mathbf{x}) = \sum_{i=1}^{k} P(i)\, Pr(\mathbf{x}|i)\,.$$

$P(i)$ are the *mixing parameters* of the model with

$$\sum_{i=1}^{k} P(i) = 1 \qquad \text{and} \qquad P(i) > 0 \quad \text{for} \quad i = 1 \ldots k\,.$$

The probability of a data point $\mathbf{x}$ belonging to cluster i can then be expressed using Bayes' theorem:

$$Pr(i|\mathbf{x}) = \frac{Pr(\mathbf{x}|i)P(i)}{Pr(\mathbf{x})}\,. \tag{5.2}$$

Various methods have been developed for estimating the parameters of a Gaussian mixture model from a data set. The *EM (Expectation-Maximisation)* algorithm (Dempster et al., 1977) is one of the most effective methods based on maximising the likelihood $\mathcal{L}$ of the parameters given a data set $X = \{\mathbf{x}_1 \ldots \mathbf{x}_N\}$. For the Gaussian components given in Equation (5.1), the negative log-likelihood of a data set is given by

$$E = -\ln \mathcal{L} = -\sum_{n=1}^{N} \ln Pr(\mathbf{x}_n) = -\sum_{n=1}^{N} \ln\Big\{\sum_{i=1}^{k} Pr(\mathbf{x}_n|i)\, P(i)\Big\}\,.$$

E can be regarded as an error function and maximising $\mathcal{L}$ is then equivalent to minimising E (Bishop, 1995). This can be done in an iterative process. Starting with some initially guessed parameters for the mixture model, an iteration of the EM algorithm provides new parameter estimates which are proven not to increase the negative log-likelihood of the model (Dempster et al., 1977). The change in the error function from iteration t to iteration $t+1$ can be written as

$$E(t+1) - E(t) = -\sum_{n=1}^{N} \ln\left(\frac{Pr^{t+1}(\mathbf{x}_n)}{Pr^{t}(\mathbf{x}_n)}\right)$$

where $Pr^{t+1}(\mathbf{x})$ and $Pr^{t}(\mathbf{x})$ denote the probability densities evaluated using the new and the old parameters, respectively. The mixture model parameters are updated as follows[1] for $i = 1 \ldots k$:

$$\begin{aligned} P(i)^{t+1} &= \frac{1}{N}\sum_{n=1}^{N} Pr^{t}(i|\mathbf{x}_n) \\ \boldsymbol{\mu}_i^{t+1} &= \frac{\sum_{n=1}^{N} \mathbf{x}_n Pr^{t}(i|\mathbf{x}_n)}{\sum_{n=1}^{N} Pr^{t}(i|\mathbf{x}_n)} \\ \boldsymbol{\Sigma}_i^{t+1} &= \frac{\sum_{n=1}^{N} Pr^{t}(i|\mathbf{x}_n)(\mathbf{x}_n - \boldsymbol{\mu}_i^{t+1})(\mathbf{x}_n - \boldsymbol{\mu}_i^{t+1})^T}{\sum_{n=1}^{N} Pr^{t}(i|\mathbf{x}_n)} . \end{aligned}$$

The membership probability of a data point $\mathbf{x}$ in each cluster can then be calculated using Equation (5.2). The process is repeated until $E(t+1) - E(t) \leq \epsilon$.

The EM algorithm is closely related to the commonly used *K-Means* algorithm (MacQueen, 1967) which assigns each data point to exactly one cluster and minimises the sum of squares of Euclidean distances between the data points in a cluster and the cluster mean vector $\boldsymbol{\mu}$. Suppose spherical Gaussians with a common width parameter σ, then for $\sigma \to 0$ we get

$$\frac{Pr(i|\mathbf{x}_n)}{Pr(j|\mathbf{x}_n)} \to 0 \quad \text{if} \quad \|(\mathbf{x}_n - \boldsymbol{\mu}_i)(\mathbf{x}_n - \boldsymbol{\mu}_i)^T\| > \|(\mathbf{x}_n - \boldsymbol{\mu}_j)(\mathbf{x}_n - \boldsymbol{\mu}_j)^T\| .$$

Thus, the probability of the data point $\mathbf{x}_n$ belonging to a cluster is zero except for the cluster i whose centre $\boldsymbol{\mu}_i$ is closest to $\mathbf{x}_n$, and the EM update formula reduces to the k-means update formula (Bishop, 1995).

[1] The update equations are obtained by setting the partial derivatives of E with respect to the individual parameters to zero. For details see, e.g., Bishop (1995).

Note finally that in order to avoid a large number of local maxima in the likelihood function, the EM algorithm can be combined with a split-and-merge method. Local maxima arise when there are too many components of a mixture in one part of the space and too few in another (Ueda et al., 2000). After performing the original EM algorithm until convergence, clusters can be split and merged according to some performance criteria. Two clusters i and j can be merged, for example, if $Pr(i|\mathbf{x}) \approx Pr(j|\mathbf{x})$ for a large number of data points. For details on possible merging and splitting criteria used in combination with the EM algorithm see, e.g., Ueda et al. (2000).

5.2.3 Connectionist Networks for Clustering

Probably the most commonly used method for clustering data using connectionist networks is *unsupervised competitive learning*. In a connectionist network a number k of competing units represent the k clusters. This layer of units is fully connected with the input layer of the network. Thus, each unit receives n weighted signals from the input units, where the data to be clustered is represented by a set $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$ of n-dimensional normalised vectors. The network is trained using the following learning algorithm.

Competitive Learning (after Rojas (1996) and Hecht-Nielsen (1990))

1. Generate n-dimensional normalised weight vectors $\mathbf{w}_1 \ldots \mathbf{w}_k$ randomly.

2. Select an input vector $\mathbf{x} \in X$ at random.
   Compute $\mathbf{x} \cdot \mathbf{w}_i$ for $i = 1 \ldots k$.
   Select $\mathbf{w}_b$ such that $\mathbf{x} \cdot \mathbf{w}_b \geq \mathbf{x} \cdot \mathbf{w}_i$ for all $i = 1 \ldots k$.

3. Update $\mathbf{w}_b$, normalise, and continue with step 2.

The first step of the algorithm generates the initial cluster centroids represented by one unit each. In the second step the unit closest to the input vector $\mathbf{x}$ is located. Since the input and weight vectors are normalised, the dot products $\mathbf{x} \cdot \mathbf{w}_i$ represent the similarity of the input vector to all weight vectors. The weight vector $\mathbf{w}_b$ producing the highest dot product is the one corresponding to the unit closest to the input vector.

This is the only weight vector that is updated in step 3. Different updating rules can be used. In the simplest case the vector $\mathbf{x}$ multiplied with a learning rate η is added to the weight vector:

$$\mathbf{w}_b^{new} = \mathbf{w}_b^{old} + \eta\,\mathbf{x}\,.$$

More often, the *Kohonen learning rule* is used which moves the weight vector towards the input vector by a fraction of the difference between the two:

$$\mathbf{w}_b^{new} = \mathbf{w}_b^{old} + \eta\,(\mathbf{x} - \mathbf{w}_b^{old})\,.$$

In both cases, only unit b is moved towards $\mathbf{x}$ and the weight vectors of all units other than b stay unchanged, a strategy of weight updating which is often referred to as *winner-takes-it-all.* Note that Kohonen learning is very similar to k-means clustering where clusters are chosen such that the sum over the distances between the input vectors and the k cluster centroids is minimised (Hecht-Nielsen, 1990).

One problem unsupervised competitive learning shares with other non-hierarchical clustering methods is that the number of clusters, i.e., the number of competing units in the network, has to be chosen in advance. Moreover, different random initialisations can lead to very different results. Fritzke (1997) observes that the purely local adaptation of weights does not guarantee to get out of a poor local minimum where the system might have started in. This problem can be overcome, however, by abandoning the winner-takes-it-all strategy and allowing weight changes for more than one unit. This process, often referred to as *self-organisation,* is realised in another network architecture making use of the Kohonen learning rule, the *Self-Organising Map* (SOM) (Kohonen, 1982).

A SOM is a connectionist network that learns the topological mapping of a n-dimensional space onto a (typically) 2-dimensional rectangular array of k units by means of self-organisation. Like in the competitive learning algorithm above, the unit b that is closest to a given input vector $\mathbf{x}$ is located and moved towards the input vector by updating its weight vector $\mathbf{w}_b$. In addition, the weight vectors of the topological neighbours of unit b are updated such that they move towards the vector $\mathbf{x}$, although to a lesser extent. Thus, an updating rule for weight vectors could be written as

$$\mathbf{w}_i^{new} = \mathbf{w}_i^{old} + \eta\,\phi(i,b)\,(\mathbf{x} - \mathbf{w}_i^{old}) \qquad \text{for } i = 1 \ldots k\,.$$

$\phi(i, b)$ is a neighbourhood function representing the strength of the coupling of unit i to unit b and η is the learning rate. $\phi(i, b) = 0$ for all units i that are not topological neighbours of b. Thus, the weight vectors of these units stay unchanged. Both the value of ϕ and η are reduced gradually during training.

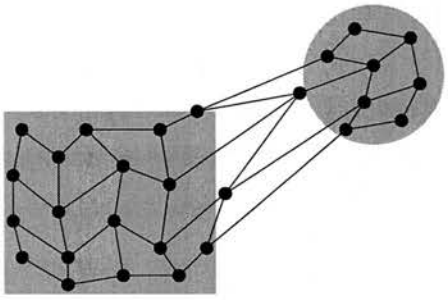

Figure 5.2: A 4×7 SOM after training with input vectors uniformly distributed in the shaded areas. The locations of the units indicate the two clusters in the input space.

Note that SOMs do not explicitly cluster the input data provided but learn the topology of the input space. However, if the input vectors used for training the SOM form natural clusters in the representational space, these clusters will be represented in the result of the learning process as illustrated in Figure 5.2.

5.2.4 Growing Neural Gas

SOMs inspired the development of the network architecture we used for clustering HRR vectors, *Growing Neural Gas* (GNG) (Fritzke, 1995). There are at least two problems with SOMs. As mentioned before, the size of the network has to be specified in advance and is fixed. In addition, the predefined structure of the network (mostly rectangular) may not always be suitable to represent the given inputs (Fritzke, 1999). The GNG network is a constructivist network architecture that is able to learn the topology of its input data and that overcomes these limitations. *Constructivism* is a theory of knowledge acquisition based on the assumption that learning itself creates and modifies the architecture of a learning system (e.g., Piaget (1955); von Foerster (1973); von Glasersfeld (1984)). Applied to connectionist networks, constructivism allows the modification and extension of the network architecture during learning. For recent developments of constructivist neural networks see, for example, Westermann

(2000).

Starting with two connected units, a GNG network learns the topology of the input data by constructing a layer of units. Like in the SOM, when an input vector $\mathbf{x}$ is presented in the input layer of the network, the unit b closest to $\mathbf{x}$ is moved towards $\mathbf{x}$ by updating its weight vector:

$$\mathbf{w}_b^{new} = \mathbf{w}_b^{old} + \eta_1(\mathbf{x} - \mathbf{w}_b^{old}) .$$

Weight vectors of all topological neighbours of unit b are updated by

$$\mathbf{w}_i^{new} = \mathbf{w}_i^{old} + \eta_2(\mathbf{x} - \mathbf{w}_i^{old})$$

with $\eta_1 > \eta_2$. Further, the distance of unit b to the vector $\mathbf{x}$ is added to a local error counter for unit b. Such an error counter is associated with each unit and is used to determine where new units should be added to the network. After a predefined number of cycles through the training data, a new unit is added halfway between the unit with the highest error and its topological neighbour with the highest error amongst all neighbours. A high error counter for a unit indicates that either

- the unit covers a large area of the input space or
- the unit lies in an area with very dense input data.

In the first case the unit accumulates a high error because it is the closest unit to a number of input vectors with a relatively large distance. In the second case the unit is the closest unit to a large number of very close input vectors. In both cases, inserting a unit in this area results in a new network topology that better matches the input data.

Finally, the network needs to construct connections representing the neighbourhood relations between units. For each input vector $\mathbf{x}$ a connection is inserted (if it does not already exist) between the two units closest and second-closest to $\mathbf{x}$. Edges no longer necessary due to the unit movement and the insertion of new units are removed by an edge aging process.

If the input data presented to the network forms clusters in the representational space, sufficient training will cause the formation of groups of connected units each representing a cluster. This is exemplified in Figure 5.3, where a network was trained

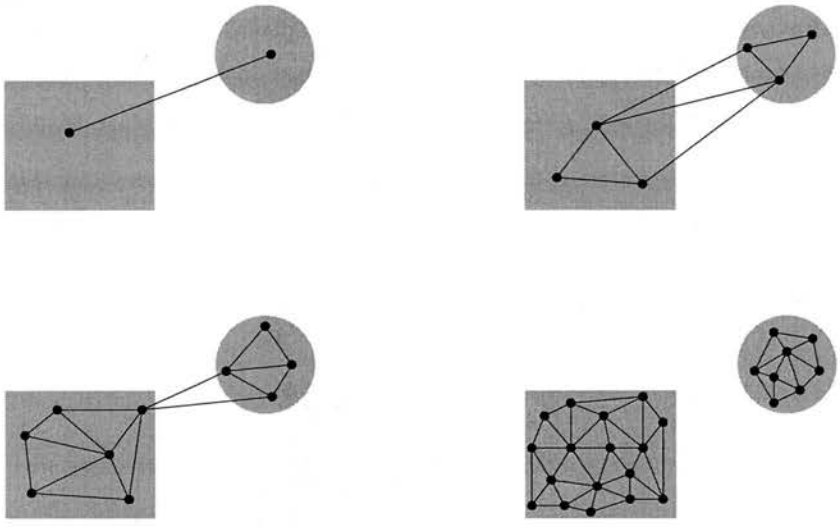

Figure 5.3: A GNG network with 2, 6, 10, and 25 units learning the topology of the input data. The input data is uniformly distributed in the shaded area.

with input vectors drawn from the distribution indicated by the grey rectangle and circle.

One problem the GNG network shares with non-hierarchical clustering methods is finding the optimal number of clusters to best show the grouping of the input data, although here we do not have to determine the number of clusters in advance. However, if training is not terminated at a certain point, the network can potentially grow until each data point is assigned exactly one unit. Possible stopping criteria for training are a maximum number of units or some performance criterion such as the maximum distance of any data point to its closest unit (Fritzke, 1995). In our implementation we let the network grow to include (at least) as many units as clusters expected. In this way, each unit represents one cluster centroid. The distances between the cluster centroids, between cluster members and the respective cluster centroid, and between cluster centroids and members of other clusters were used to verify that detected clusters were well separated.

Note that GNG implements aspects of both hierarchical and non-hierarchical clustering. Like in hierarchical divisive clustering the number of clusters is increased successively. However, the assignment of an input vector to a cluster does not depend on its assignment in earlier stages. Thus, each step of the learning process can also

be interpreted as an independent non-hierarchical clustering with the cluster centroids obtained in the previous step as initialisation.

5.2.5 Principal Component Analysis

A number of researchers have used Principal Component Analysis (PCA) for the detection of clusters in high-dimensional data such as the representational spaces of RAAMs. PCA is not so much a clustering technique as a statistical method used to reduce the dimensionality of given data with minimal loss of information. For a given set $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$ of n-dimensional input vectors PCA performs a linear transformation of X into a new set $Y = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_m\}$ of p-dimensional vectors. $\mathbf{y}_i$ and $\mathbf{y}_j$ are uncorrelated for all $i \neq j$ with $i, j = 1 \ldots m$ and $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_m$ account for decreasing portions of the variance of the original vectors. The transformation matrix is the matrix of eigenvectors of the covariance matrix of X, where eigenvectors are ordered in descending order of the magnitude of their corresponding eigenvalues. Eigenvectors corresponding to eigenvalues with a very small magnitude are usually discarded which leads to the dimensionality reduction in the new representational space ($p < n$). Plotting pairs of principal component scores often allows for the visual inspection of the structure present in the original data set[2]. Thus PCA is often used to detect clusters in a data set. As Everitt (1993) points out, however, low-dimensional principal component plots can sometimes conceal the clusters present in the data.

Note that the principal components of a data set can also be learned using unsupervised learning in a network of units with linear activation function (Rojas, 1996). In such a network, the first principal component of a data set $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$ is the weight vector $\mathbf{w}$ that maximises

$$\frac{1}{m} \sum_{i=1}^{m} ||\mathbf{w} \cdot \mathbf{x}_i||^2 .$$

Further principal components are computed from the first one in a recursive manner. For simplicity, we used the method of calculating the eigenvectors of the covariance matrix in our own analysis of the representational space of RAAM structures.

[2]A plot of a pair of principal components describes a projection of the data points in the high-dimensional space onto a plane.

5.3 Spatial Structure of RAAM Representations

As observed in Section 5.1 some information about the structure of hierarchical objects represented in RAAMs can be encoded locally in the hidden units of the RAAM. Most structural information will be distributed over the entire hidden layer, however, in particular if the amount of structural information about an object exceeds the number of units. In this case, clustering can reveal the spatial structure of the learned representations.

5.3.1 Previous Results

A large amount of analytical work has already been done regarding the representation of hierarchical objects formed in the hidden layers of RAAMs. Pollack (1988) applied a hierarchical cluster analysis to the RAAM representations of well-formed syntactic trees. The resulting clusters reflected the depth of trees and what type of phrase a tree represented; sentences were well separated from non-sentences, verb phrases from noun phrases, and so on.

Niklasson and Bodén (1997) applied a hierarchical cluster analysis to RAAM representations for binary trees. As we have seen in Section 3.2.1, a single perceptron was able to classify these representations according to the depth of the tree and the balance of the tree. The cluster analysis applied to the underlying RAAM representations detected three types of clusters:

1. one cluster of well-balanced trees,
2. clusters of left-balanced trees having a particular element as the right-most leaf,
3. clusters of right-balanced trees having a particular element as the left-most leaf.

This clustering does not come as a complete surprise, if we bear in mind the general shape of the trees and the way the trees were formed in the RAAM. In the particular tree representations used, the right-most leaf in a left-balanced tree was the element closest to the root. This was also the last element to be fed into the RAAM when it was trained to encode and decode the structures. The same holds for the left-most leaf in a right-balanced tree. We would therefore expect the elements in these two positions to

have a larger influence on the resulting representation than elements lower down in the hierarchy.

This general form of clustering was further observed by Stolcke and Wu (1992) who applied a hierarchical cluster analysis to the representations of binary trees that formed the basis for the holistic matching of trees discussed in Section 3.2.2. The results of the clustering show "the primary factor [for the clustering] to be tree structure. Within each set of identically-structured trees, the values of the terminals govern the sub-clustering, with terminals closer to the root being more important than deeper terminals (Stolcke and Wu, 1992, p. 4)."

It turns out that this form of clustering can naturally support certain holistic operations. For example, looking back at Niklasson and Bodén's tasks, the classification of trees according to their balance and according to their depth, the observed clusters clearly provided the information for a classifier perceptron to distinguish left-balanced, right-balanced and well-balanced trees. Because of the way the trees were presented to the RAAM, the clustering coincided with the correct distinction of the trees required to learn this classification problem. The different clusters did not govern any information of the tree depth, however, which was required for the second classification task. There were left- and right-balanced trees of depth 3 and trees of depth 2 with all possible balances. However, as observed in Section 5.1 a single hidden unit encoded for the tree depth. This local encoding provided the depth classifier with the relevant information to correctly separate trees of depth 2 from trees of depth 3 without taking the clusters formed in the representational space into account. Both holistic classifications of the RAAM structures could thus be learned by simple independently trained perceptrons.

Sperduti et al. (1995) used PCA to investigate the difference between the sequential and the parallel training of a RAAM and a network learning the holistic classification of the RAAM structures. PCA was applied to representations of logical terms formed in a LRAAM combined with a classification perceptron (cf. Section 3.2.1). LRAAM and classifier were trained, first separately then in parallel, to encode and decode different logical expressions and then detect expressions of a particular form, such as expression of the form $f(X, X)$. When LRAAM and classification network were trained independently, the cluster analysis showed that the representations for expressions were formed such that expressions of the same depth fell into the same

cluster, whereas expressions of different depths formed separate clusters. It was not the depth of the expressions but instances of a particular form the classification network was supposed to detect, however. These clusters therefore did not provide the information necessary for the classification task. As reported in Section 3.2.1, an independently trained classification network was indeed not able to correctly classify these representations.

The cluster analysis was repeated for the parallel architecture. Now, the representations in the hidden layer of the LRAAM were formed under the influence of both the encoding/decoding task of the LRAAM and the classification task at hand. As a result of the parallel training the representations were now clustered with respect to the classification task. Positive examples for the classification formed clusters well separated from clusters containing negative examples. For some classification tasks, the first and second principal component or even only the first one were sufficient to separate the representations (Sperduti et al., 1995).

These results explain the huge performance increase that is observed for learning complex holistic operations on RAAM structures, when an interleaved training of both networks is performed. Representations formed in a separately trained RAAM do not always form clusters supporting the subsequently applied holistic operations. However, the parallel training of both networks modifies the spatial structure of the representations as to capture the structural information necessary for the holistic operation at hand.

Niklasson (1999) observed that using a special bit in RAAM representations dedicated to the distinction between elements and composed structures enhances the encoding/decoding performance of an RAAM. As noted in Section 2.3, using this explicit distinction eases the decision in the decoding process about whether a vector represents an atomic element or needs further decoding. However, it also results in more accurately decoded element vectors. Again a cluster analysis of the representations formed in the hidden layer of a RAAM provides some explanation for the difference between the presence and the absence of this bit. A dual RAAM was trained to encode trees of the forms seen in Figure 5.4 with the two elements A and B as leaves. After training the RAAM without the explicit distinction between elements and structures, the resulting representations formed 4 clusters in the representational

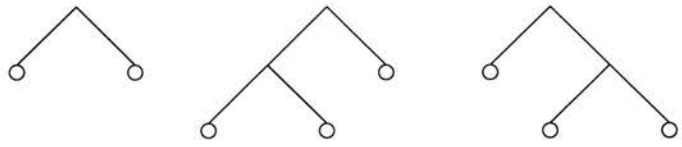

Figure 5.4: Different tree structures encoded by dual RAAMs with and without explicit element/structure classification. Leaves could take on the two values A and B.

space: $(A(x_1, x_2))$, $(B(x_1, x_2))$, $((x_1, x_2)A)$ and $((x_1, x_2)B)$ with $x_1, x_2 \in \{A, B\}$. The four well-balanced trees were placed separately between these clusters. Training was performed again, now using the extra bit distinguishing an element from a structure. The cluster analysis resulted in the same but tighter and better separated clusters. Moreover, the clusters were placed differently in the representational space such that clusters containing trees with the same balance were closer to each other than to clusters of trees with a different balance. Finally, the well-balanced trees were placed more systematically in the representational space, i.e., closer to each other and better separated from left- and right-balanced trees. In short, the extra bit resulted in a more systematically structured representational space, which in turn allows for a more accurate decoding of the structure elements.

5.3.2 Our Own Experiments

So far we have seen that the well structured representational space of learned RAAM representations enables a connectionist network to learn the holistic classification and matching of the representations formed in the RAAM. With our own experiments we demonstrate that a well clustered representational space also forms the basis for holistic transformations of RAAM representations. We applied PCA to the representations formed in a ternary RAAM for Niklasson's (1993) transformation task

$$(x_1 \; op \; x_2) \rightarrow x_3 \iff \neg (x_1 \; op \; x_2) \vee x_3$$
$$x_1 \rightarrow (x_2 \; op \; x_3) \iff \neg x_1 \vee (x_2 \; op \; x_3)$$

with $x_1, x_2, x_3 \in \{p, q, r, s\}$ and $op \in \{\rightarrow, \vee\}$. The representation of the elements, the network architectures, and the results of training and testing RAAM and TN were discussed in detail in Section 3.2.3. Recall that representations used for this task were transformed and decoded by fully trained RAAM and TN with 94% accuracy.

PCA was first applied to the input structures for the transformations formed in the hidden layer of the RAAM. Figure 5.5 shows the first principal component plotted against the second principal component. It can clearly be seen that the representations form clusters in the representational space that distinguish disjunctions from implications and structures with a complex first and atomic second expression from structures with a complex second and atomic first expression. This distinction provides enough information for the transformation network to apply the appropriate part of the learned transformation which consists of both the transformation of a disjunction into an implication and its reverse.

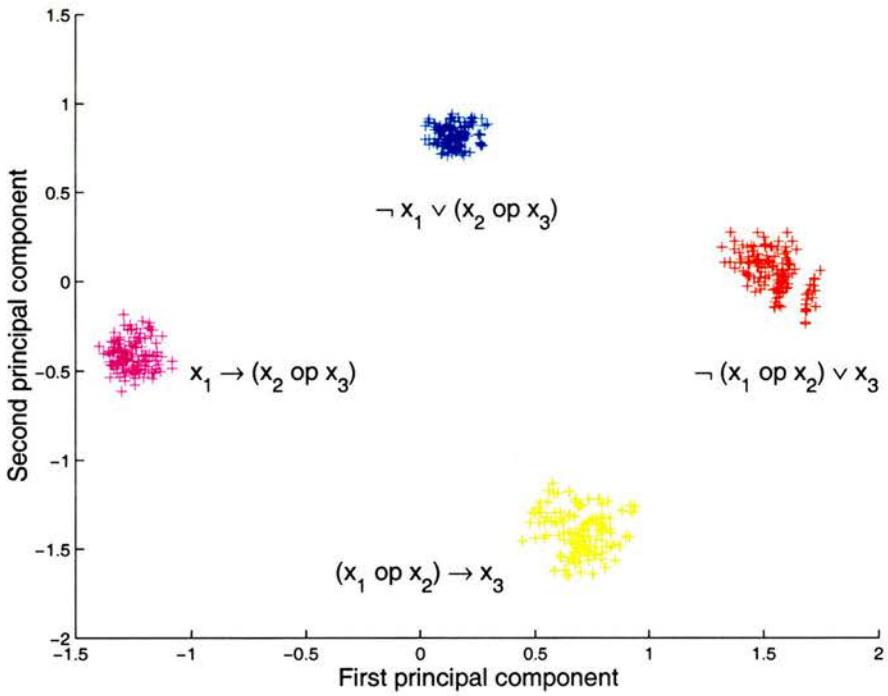

Figure 5.5: First and second principal component of four classes of structures represented by a ternary RAAM. The classes comprise the input data for a holistic transformation with $x_1, x_2, x_3 \in \{p, q, r, s\}$ and $op \in \{\rightarrow, \vee\}$.

In order to show the influence of a holistic transformation on the representational space of the structures, we applied PCA to the results of the transformation above. Figure 5.6 shows the results of the transformation after the TN was trained on the whole data set. Again, the first principal component is plotted against the second principal component. We further kept the same colour coding for classes of structures

as in Figure 5.5, i.e., the class of input structures of the form $\neg x_1 \vee (x_2\ op\ x_3)$ (blue in Figure 5.5) is now transformed into the class of the form $x_1 \rightarrow (x_2\ op\ x_3)$ (also blue in Figure 5.6), and so on. The four clusters representing the transformed structures are still very clearly separated. However, the transformed structures cluster with more variance than the input structures of the transformation, which can be seen in particular for the clusters representing the input class $\neg x_1 \vee (x_2\ op\ x_3)$ and the transformed class $x_1 \rightarrow (x_2\ op\ x_3)$ (both blue in Figure 5.5 and Figure 5.6, respectively). This larger variance in the transformed structures can be explained by the noise introduced during the transformation.

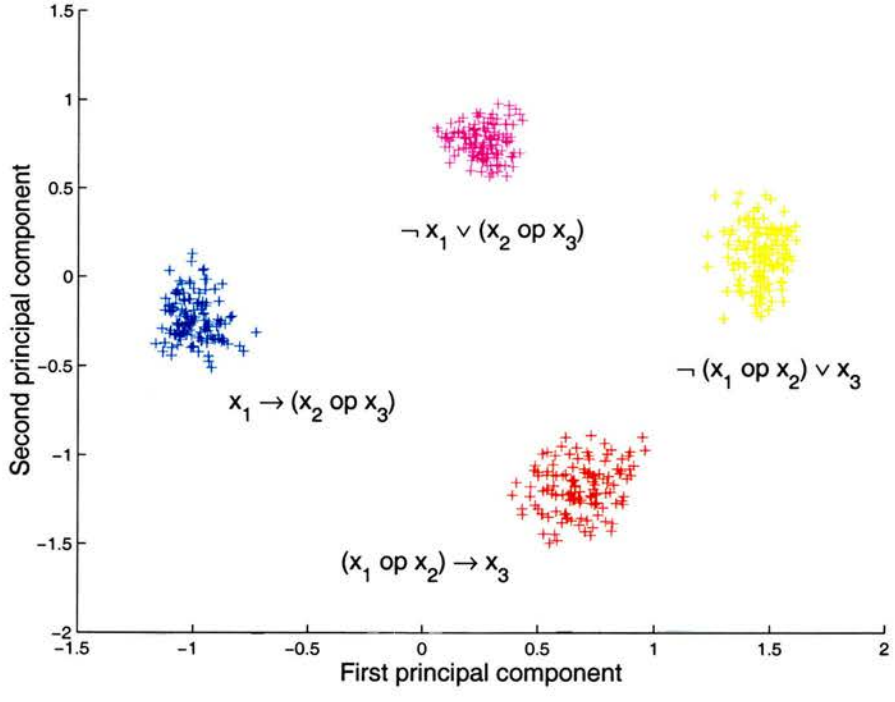

Figure 5.6: First and second principal component of four classes of structures. The structures were the result of a holistic transformation by a TN that was trained on all structures with $x_1, x_2, x_3 \in \{p, q, r, s\}$ and $op \in \{\rightarrow, \vee\}$.

The cluster analysis was repeated after training the TN only on structures not containing the element s and testing it on the whole data set. Recall from Section 3.2.3 that a fully trained RAAM was not able to correctly decode the transformed structures of the test set containing the element s in any position in the structure. 19.5% of the transformed structures were wrongly decoded and most decoding error were observed

for the previously unseen element s.

Figure 5.7 shows the first principal component of the test set plotted against the second. Structures containing the element s are represented by black circles. Somewhat surprisingly, the four clusters representing the four classes of structures are again clearly separated and all structures containing the element s fall into the appropriate clusters. This result does not significantly differ from the clustering obtained from

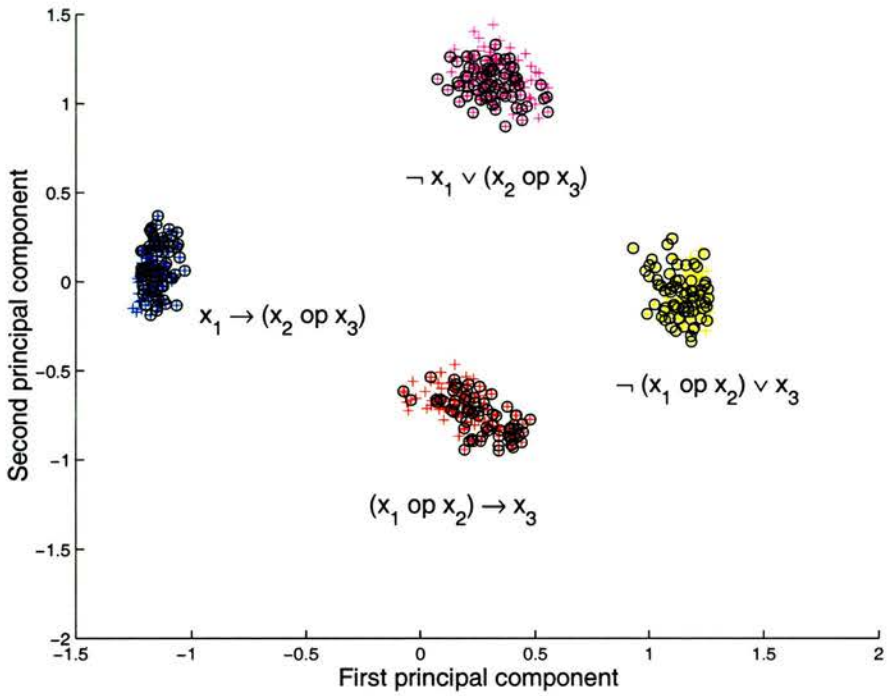

Figure 5.7: First and second principal component of four classes of structures. The structures were the result of a holistic transformation by a TN that was trained on structures not containing the element s. The transformed structures of the test set containing the element s are represented by black circles.

a fully trained TN and thus does not provide an explanation for the failure of the TN to generalise to novel elements in the structures. However, we found that the fourth principal component of the representations seems to account for the variance of the structures within each cluster. Figure 5.8 shows, for example, the clustering within the class of the form $\neg\,(x_1\ op\ x_2) \vee x_3$. The input to a TN $(+)$, the transformed structures obtained from a fully trained TN $(\circ)$, and the transformed structures from a TN trained without the element s $(*)$ are shown. The input structures as well as the result struc-

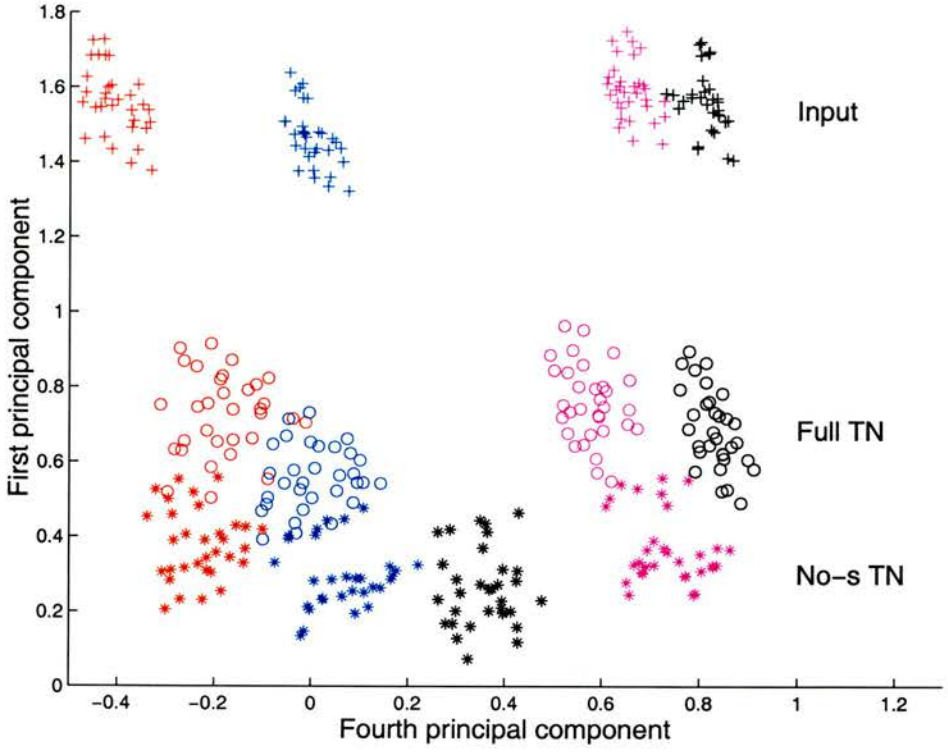

Figure 5.8: First and fourth principal component of a single class of structures. The input data of a TN, the transformed structures from a fully trained TN, and the transformed structures from a TN trained without the element s are shown. Within each set, four clusters can be identified containing the structures $\neg\,(x_1\ op\ x_2) \vee p$ (red), $\neg\,(x_1\ op\ x_2) \vee q$ (blue), $\neg\,(x_1\ op\ x_2) \vee r$ (pink), and $\neg\,(x_1\ op\ x_2) \vee s$ (black).

tures of both transformations are clustered into four groups (read, blue, pink, black) with respect to the instance of x_3 in the expressions. However, the picture shows a significant difference between the results of the two transformations. The fully trained TN places the cluster of transformed structures with $x_3 = s$ (black) in the same region as the untransformed input structures with $x_3 = s$, with respect to the fourth principal component. The TN trained without the element s places these structures in a different area. The structures take on positions in an area which is close to the average of the positions of transformed structures with $x_3 \in \{p, q, r\}$. The transformation network does not generalise the knowledge about the position x_3 in a structure, but rather about the elements p, q, and r found in this position during training. This confirms the observation reported in Section 3.2.3 that vectors supposed to represent the element

s after the transformation were often wrongly decoded as something representing the type of the elements p, q, and r. Similar results were observed for other classes of structures such as $(x_1\ op\ x_2) \rightarrow x_3$.

Although these wrongly placed structures comprised a large part of the structures that could not be correctly decoded by a fully trained RAAM after the transformation, the observed clustering still does not give a full explanation for the inability of the RAAM to decode *almost all* vectors representing the novel element s. An explanation for the incorrect decoding of the element s as an instance of a deeper embedded variable, i.e., $x_1 = s$ and/or $x_2 = s$ in the class of structures above, would require an even more detailed clustering of the representations used for the transformation. For example, we would expect a clustering with respect to a particular instance of x_1 or x_2 within groups of structures like $\neg\,(x_1\ op\ x_2) \vee p$ or $\neg\,(x_1\ op\ x_2) \vee q$. However, such a fine-grained clustering could not be observed using PCA.

5.4 Spatial Structure of HRR

PCA applied to structures represented by HRRs did not show the detailed clustering as observed for RAAM structures. For both the simple transformation task

$$x_1 \rightarrow x_2 \Longleftrightarrow \neg\, x_1 \vee x_2$$

and the transformation of more complex structures

$$\begin{aligned}(x_1\ op\ x_2) \rightarrow x_3 &\Longleftrightarrow \neg\,(x_1\ op\ x_2) \vee x_3\\ x_1 \rightarrow (x_2\ op\ x_3) &\Longleftrightarrow \neg\, x_1 \vee (x_2\ op\ x_3)\end{aligned}$$

all input structures were clearly divided into having a disjunction or an implication as the top-level operation, but no information could be gained about the structure of the data within these two groups. In several simulations we included the first 15 to 20 principal components into the analysis. These corresponded to eigenvalues not smaller than 0.01. All following eigenvalues were smaller than 10^{-6}. However, a more detailed clustering of HRR structures could be obtained when using a different clustering method, the constructivist GNG network. Below we report the results observed for the same element representations as used in the simulations presented in Chapter 4. Note,

however, that very similar results were observed in several replications with different randomly generated element vectors.

5.4.1 Clustering of Transformation Inputs

We first applied the clustering GNG network to the HRRs representing the structures $x_1 \rightarrow x_2$ and $\neg x_1 \vee x_2$ with $x_1, x_2 \in \{p, q, r, s\}$. From the initial results using PCA we expected the network to divide the whole data set into two clusters containing the implications and disjunctions, respectively. Indeed, after the two randomly placed initial units were trained for 10 iterations through the data set, these two clusters emerged. The Euclidean distance between the cluster centroids was 2.21, the maximum distance of all data points to their respective cluster centroid was 1.84, and the minimum distance of all data points to other cluster centroids was 2.30. When more units were added to the network, the two initial clusters were further split up. After constructing 8 units, the network clustered the input data as shown in Table 5.1. New units were constructed after every 10 cycles though all training examples. The minimum Euclidean distance between the new cluster centroids was 1.40, the maximum distance of a data point to its own cluster centroid was 1.22, and the minimum distance of all data points to other cluster centroids was 1.44.

Cluster	Cluster members				Characteristic
1	$\neg p \vee s$	$\neg q \vee s$	$\neg r \vee s$	$\neg s \vee s$	$x_1 \vee s$
2	$p \rightarrow s$	$q \rightarrow s$	$r \rightarrow s$	$s \rightarrow s$	$x_1 \rightarrow s$
3	$\neg p \vee q$	$\neg q \vee q$	$\neg r \vee q$	$\neg s \vee q$	$x_1 \vee q$
4	$p \rightarrow q$	$q \rightarrow q$	$r \rightarrow q$	$s \rightarrow q$	$x_1 \rightarrow q$
5	$\neg p \vee p$	$\neg q \vee p$	$\neg r \vee p$	$\neg s \vee p$	$x_1 \vee p$
6	$p \rightarrow p$	$q \rightarrow p$	$r \rightarrow p$	$s \rightarrow p$	$x_1 \rightarrow p$
7	$\neg p \vee r$	$\neg q \vee r$	$\neg r \vee r$	$\neg s \vee r$	$x_1 \vee r$
8	$p \rightarrow r$	$q \rightarrow r$	$r \rightarrow r$	$s \rightarrow r$	$x_1 \rightarrow r$

Table 5.1: Clusters formed after constructing a layer of 8 units in a GNG network for HRRs representing the structures $x_1 \rightarrow x_2$ and $\neg x_1 \vee x_2$ with $x_1, x_2 \in \{p, q, r, s\}$. New units were added every 10 cycles through all training examples. The structures are clustered with respect to the operation and its second constituent.

We repeated the clustering with the same input data but 100 cycles through the training set before new units were added. The results of the clustering are presented in Table 5.2. The minimum distance between the cluster centroids was 1.41, the maximum distance of a data point to its own cluster centroid was 1.23, and the minimum distance of all data points to other cluster centroids was 1.38.

Cluster	Cluster members				Characteristic
1	$\neg r \vee p$	$\neg r \vee q$	$\neg r \vee r$	$\neg r \vee s$	$\neg r \vee x_2$
2	$r \rightarrow p$	$r \rightarrow q$	$r \rightarrow r$	$r \rightarrow s$	$r \rightarrow x_2$
3	$\neg q \vee p$	$\neg q \vee q$	$\neg q \vee r$	$\neg q \vee s$	$\neg q \vee x_2$
4	$\neg s \vee p$	$\neg s \vee q$	$\neg s \vee r$	$\neg s \vee s$	$\neg s \vee x_2$
5	$p \rightarrow p$	$p \rightarrow q$	$p \rightarrow r$	$p \rightarrow s$	$p \rightarrow x_2$
6	$q \rightarrow p$	$q \rightarrow q$	$q \rightarrow r$	$q \rightarrow s$	$q \rightarrow x_2$
7	$\neg p \vee p$	$\neg p \vee q$	$\neg p \vee r$	$\neg p \vee s$	$\neg p \vee x_2$
8	$s \rightarrow p$	$s \rightarrow q$	$s \rightarrow r$	$s \rightarrow s$	$s \rightarrow x_2$

Table 5.2: Clusters formed after constructing a layer of 8 units in a GNG network for HRRs representing the structures $x_1 \rightarrow x_2$ and $\neg x_1 \vee x_2$ with $x_1, x_2 \in \{p, q, r, s\}$. New units were added every 100 cycles through all training examples. The structures are clustered with respect to the operation and its first constituent.

A comparison of Table 5.1 and Table 5.2 shows that the structures were clustered differently, once with respect to the top-level operation and its second constituent and once with respect to the top-level operation and its first constituent. This different clustering of the same data obtained from two training runs of the GNG network suggests that the structure of the representational space of HRRs is comparatively complex. This can be explained by the fact that the construction procedure for HRRs allows for a very flexible combination of elements and is not restricted to the combination of only two elements or three elements at a time which is the case for dual and ternary RAAMs, respectively. The resulting hierarchical objects viewed as trees tend to be flat and wide rather than narrow and deep.

This effect is exemplified in Figure 5.9, which compares the tree structures obtained from encoding the expression $\neg p \vee q$ using HRRs and a dual RAAM. In this example the expected Euclidean distance of two HRR vectors only differing in the

first constituent[3] is of the same magnitude as the expected difference between two representations differing in the second constituent of the expression. Consequently, the representational space can be naturally clustered with respect to both constituents in the representations. We have seen in the previous Section, however, that representations of structurally equivalent trees learned by a dual RAAM were always clustered with respect to the constituent closest to the root of the tree, the element q in our example. The distance between two representations only differing in this constituent

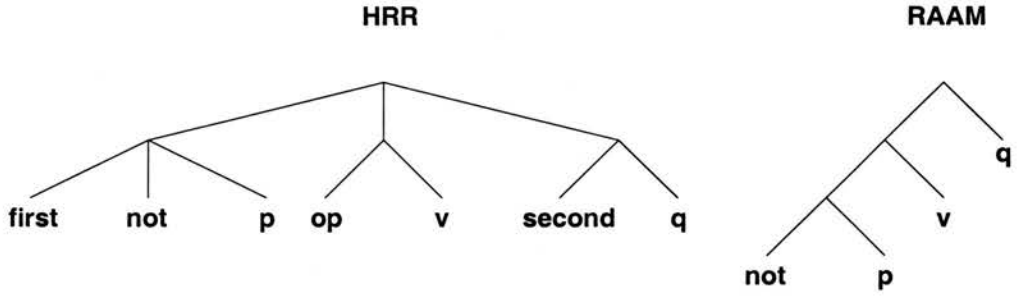

Figure 5.9: Two different hierarchical structures obtained from encoding the expression $\neg p \vee q$ using HRRs and a dual RAAM. The HRR structure was build by **not_p_or_q** = (**op** $\circledast$ **disj** + **first** $\circledast$ **not** $\circledast$ **p** + **second** $\circledast$ **q**).

is generally larger than then distance between two representations differing in only a single deeper embedded constituent, given that the overall tree structure is the same. The spatial structure of the representational space of RAAM structures is less complex. However, we would expect to see similar effects of different possible clusterings as for HRRs, when RAAMs with 3 or more input slots are used.

Given that the dimensionality of the developing GNG network is not restricted to two or three, it is somewhat difficult to visualise the detected clusters. We have chosen a pseudo-colour plot showing the distances of all data points representing structures (x-axis) to the cluster centroids obtained from the GNG network (y-axis). The scale of the distances is shown above each plot, reaching from 0 (dark blue) to the maximum distance of a data point to any cluster centroid (dark red). The clusters in the data set can be seen as groups of blue to green entries, i.e., as groups of data points that are close to a particular cluster centroid. The order of the data points on the x-axis

[3]Note, however, that the first constituent here is $\neg p$ and not the atomic element p.

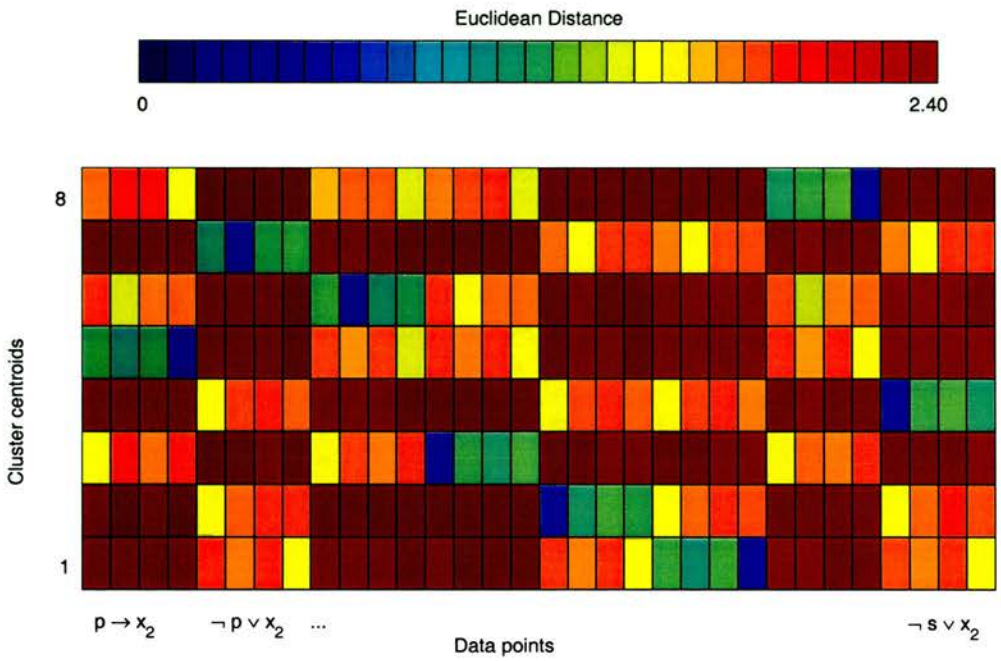

Figure 5.10: Distances between the cluster centroids and data points representing structures $x_1 \rightarrow x_2$ and $\neg x_1 \vee x_2$ with $x_1, x_2 \in \{p, q, r, s\}$. The 8 clusters seen correspond with the clusters in Table 5.2.

is indicated below the plot. Figure 5.10 shows the distances of the data points to the cluster centroids for the clusters presented in Table 5.2.

5.4.2 Generalisation to Novel Elements

In order to see how well transformed structures cluster in the representational space, especially if they contain previously unseen elements, we learned the transformation $x_1 \rightarrow x_2 \Longleftrightarrow \neg x_1 \vee x_2$ using the one-shot learning algorithm. The training set consisted of all structures with $x_1, x_2 \in \{p, q, r, s\}$. After training, structures containing the new element t were added to form the test set. The learned transformation vector transformed both training and test set with 100% accuracy. The transformed test set was then clustered using a GNG network where new units were added every 100 cycles through all examples. We expected the network to form clusters similar to the ones presented in Table 5.2 plus two new clusters containing structures of the forms $t \rightarrow x_2$ and $\neg t \vee x_2$. This hypothesis proved to be correct. The results of the clustering

are shown in Table 5.3 and Figure 5.11. As expected, two new clusters, clusters 3 and

Cluster	Cluster members					Characteristic
1	$q \to p$	$q \to q$	$q \to r$	$q \to s$	$q \to t$	$q \to x_2$
2	$r \to p$	$r \to q$	$r \to r$	$r \to s$	$r \to t$	$r \to x_2$
3	$\neg t \vee p$	$\neg t \vee q$	$\neg t \vee r$	$\neg t \vee s$	$\neg t \vee t$	$\neg t \vee x_2$
4	$t \to p$	$t \to q$	$t \to r$	$t \to s$	$t \to t$	$t \to x_2$
5	$\neg r \vee p$	$\neg r \vee q$	$\neg r \vee r$	$\neg r \vee s$	$\neg r \vee t$	$\neg r \vee x_2$
6	$\neg p \vee p$	$\neg p \vee q$	$\neg p \vee r$	$\neg p \vee s$	$\neg p \vee t$	$\neg p \vee x_2$
7	$p \to p$	$p \to q$	$p \to r$	$p \to s$	$p \to t$	$p \to x_2$
8	$s \to p$	$s \to q$	$s \to r$	$s \to s$	$s \to t$	$s \to x_2$
9	$\neg s \vee p$	$\neg s \vee q$	$\neg s \vee r$	$\neg s \vee s$	$\neg s \vee t$	$\neg s \vee x_2$
10	$\neg q \vee p$	$\neg q \vee q$	$\neg q \vee r$	$\neg q \vee s$	$\neg q \vee t$	$\neg q \vee x_2$

Table 5.3: Clusters formed after constructing a layer of 10 units in a GNG network for HRRs representing the result structures of the transformation $x_1 \to x_2 \iff \neg x_1 \vee x_2$ with $x_1, x_2 \in \{p, q, r, s, t\}$. New units were added after every 100 cycles through all training examples.

4, were added containing structures of the form $t \to x_2$ and $\neg t \vee x_2$, respectively. Further, the structures containing the novel element t in position x_2 fall into the correct clusters. In Figure 5.11 the data point 5 representing $p \to t$ falls into cluster 7 with all other structures of the form $p \to x_2$, data point 10 representing $\neg p \vee t$ falls into cluster 6 together with all other structures of the form $\neg p \vee x_2$, and so on. Like for transformed RAAM structures, the basic clustering of the representational space was preserved when the HRR vectors were transformed holistically. Now, however, structures containing novel elements were placed into the representational space by the holistic transformation such that they could be correctly decoded afterwards. The learned transformation vector generalised the knowledge acquired during training to the novel element.

The preservation of the spatial structure of representations during transformation, we would argue, allows for the application of a sequence of holistic transformations to structures represented by HRR. However, we observed that clusters of transformed structures are closer together in the representational space than clusters of correspond-

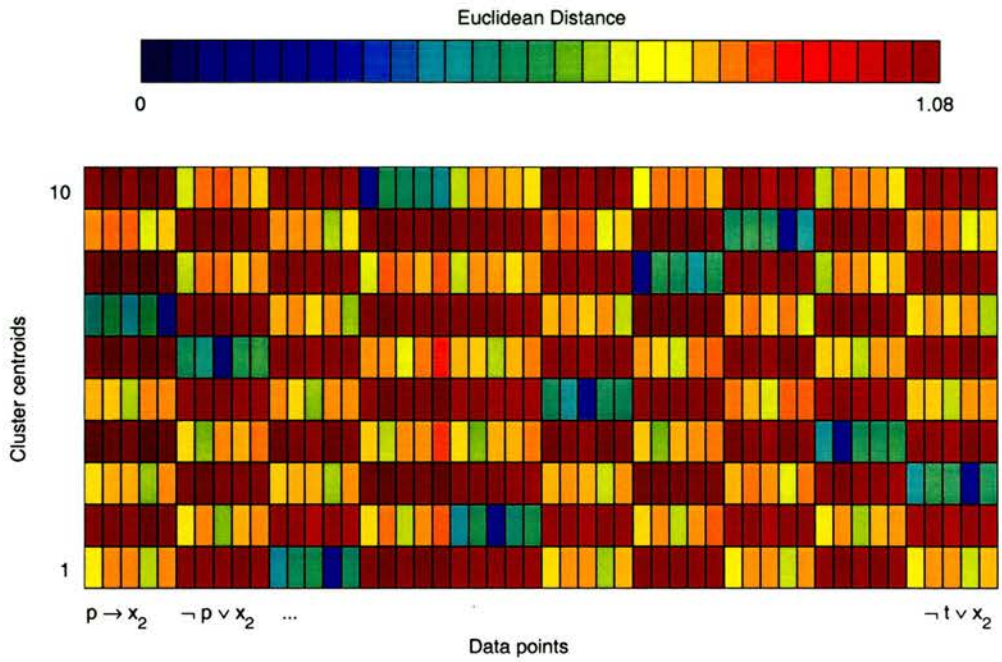

Figure 5.11: Distances between the cluster centroids and data points representing result structures of the transformations $x_1 \rightarrow x_2 \Longleftrightarrow \neg\, x_1 \vee x_2$ with $x_1, x_2 \in \{p, q, r, s, t\}$. Training was performed without the element t.

ing untransformed structures. In the example above, the distance between cluster centroids was 0.58, the maximum distance of a data point to its own cluster centroid was 0.51, and the minimum distance of all data points to other cluster centroids was 0.59. This can be attributed to the noise introduced by the transformation vector learned for a whole class of structures. The application of a long sequence of holistic transformations will therefore still require the clean-up of intermediate transformation results.

5.4.3 Generalisation to Structures of Higher Complexity

One of the main results presented in this thesis is that the algorithms developed for learning the holistic transformation of HRR structures are able to generalise acquired knowledge about simple structures to structures of higher complexity (cf. Chapter 4). A transformation vector can be learned, for example, for the transformation

$$x_1 \rightarrow x_2 \Longleftrightarrow \neg\, x_1 \vee x_2$$

and correctly generalise to the transformation

$$(x_1 \wedge x_2) \rightarrow x_3 \Longleftrightarrow \neg (x_1 \wedge x_2) \vee x_3 \,.$$

As is shown in Table 4.4 in Section 4.3 the transformation vector generalised to the structures of higher complexity with 100% accuracy. This generalisation capacity should be reflected in the spatial structure of the underlying representations. We therefore performed a cluster analysis of structures containing constituents of different complexity in the same position. A GNG network was first trained to cluster the input structures of the two transformations above: simple structures of the forms $x_1 \rightarrow x_2$ and $\neg x_1 \vee x_2$ together with the complex structures of the forms $(x_1 \wedge x_2) \rightarrow x_3$ and $\neg (x_1 \wedge x_2) \vee x_3$ with $x_1, x_2, x_3 \in \{p, q, r, s\}$. The network formed 8 clusters presented in Table 5.4. As can be seen, the structures were clustered with respect

Cluster	Cluster members	
1	$x_1 \rightarrow r$	$(x_1 \wedge x_2) \rightarrow r$
2	$\neg x_1 \vee r$	$\neg (x_1 \wedge x_2) \vee r$
3	$x_1 \rightarrow q$	$(x_1 \wedge x_2) \rightarrow q$
4	$\neg x_1 \vee s$	$\neg (x_1 \wedge x_2) \vee s$
5	$x_1 \rightarrow p$	$(x_1 \wedge x_2) \rightarrow p$
6	$\neg x_1 \vee p$	$\neg (x_1 \wedge x_2) \vee p$
7	$\neg x_1 \vee q$	$\neg (x_1 \wedge x_2) \vee q$
8	$x_1 \rightarrow s$	$(x_1 \wedge x_2) \rightarrow s$

Table 5.4: Clusters formed after constructing a layer of 8 units in a GNG network for HRRs representing the simple structures $x_1 \rightarrow x_2$ and $\neg x_1 \vee x_2$ and the complex structures $(x_1 \wedge x_2) \rightarrow x_3$ and $\neg (x_1 \wedge x_2) \vee x_3$.

to their top-level operation and its second constituent. Moreover, the corresponding groups of the simple and the complex structures were placed in the same clusters. The distances between the cluster centroids and the data points representing the structures are illustrated in Figure 5.12.

A transformation vector was then learned for the transformation of the simple structures and tested on both the simple and complex structures. Note that the connective $\wedge$ was not used during training. The transformed test structures were again clustered

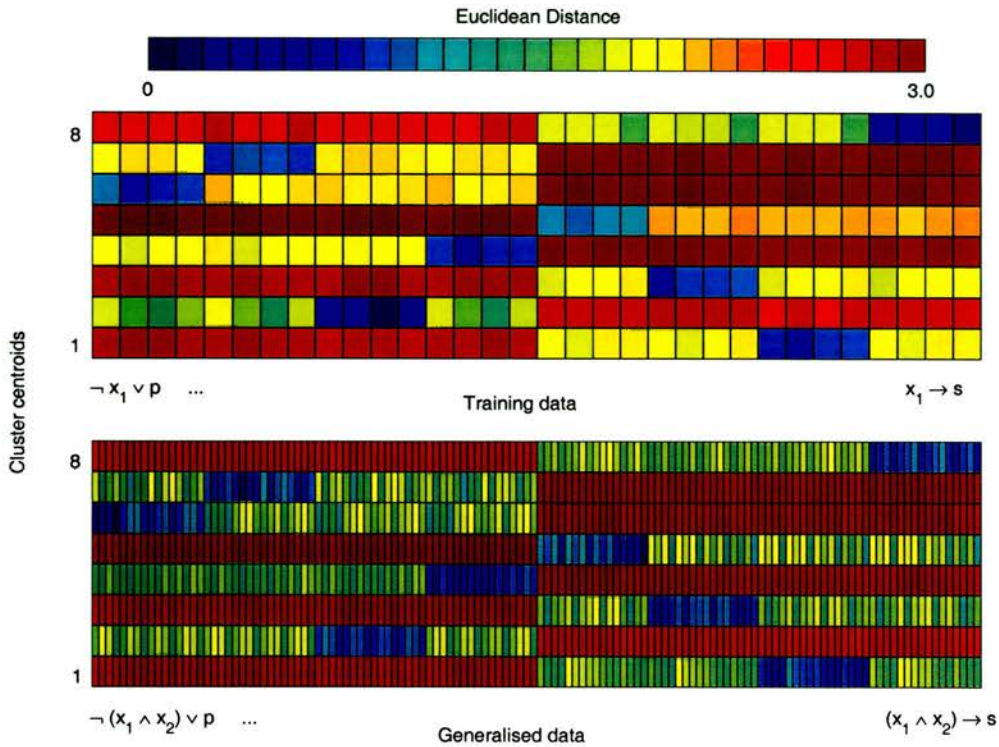

Figure 5.12: Distances between the cluster centroids and data points representing the simple structures $x_1 \rightarrow x_2$ (top right) and $\neg x_1 \vee x_2$ (top left) and the more complex structures $(x_1 \wedge x_2) \rightarrow x_3$ (bottom right) $\neg (x_1 \wedge x_2) \vee x_3$ (bottom left). Corresponding simple and complex structures fall into the same cluster.

using a GNG network. The clusters are presented in Table 5.5 and Figure 5.13. As expected, the principal spatial structure of the transformed representations is the same as that of the untransformed representations, but as observed in the previous Section, the clusters are closer together. For example, the minimum distance between two cluster centroids was 1.47 for the untransformed and 0.60 for transformed structures. A similar clustering was observed for other generalisation tasks presented in Chapter 4.

In general, the way generalisations to more complex hierarchical structures were performed, i.e., generalisations that require level 4 and level 5 systematicity, did not significantly differ from level 3 generalisations to structures containing novel elements. Representations containing novel elements were placed into clusters according to the role of the novel element in the expression. In the same way, structures containing constituents of higher complexity were placed into clusters according to the role of

Cluster	Cluster members	
1	$x_1 \rightarrow r$	$(x_1 \wedge x_2) \rightarrow r$
2	$\neg x_1 \vee r$	$\neg (x_1 \wedge x_2) \vee r$
3	$x_1 \rightarrow s$	$(x_1 \wedge x_2) \rightarrow s$
4	$\neg x_1 \vee q$	$\neg (x_1 \wedge x_2) \vee q$
5	$\neg x_1 \vee p$	$\neg (x_1 \wedge x_2) \vee p$
6	$x_1 \rightarrow q$	$(x_1 \wedge x_2) \rightarrow q$
7	$\neg x_1 \vee s$	$\neg (x_1 \wedge x_2) \vee s$
8	$x_1 \rightarrow p$	$(x_1 \wedge x_2) \rightarrow p$

Table 5.5: Clusters formed after constructing a layer of 8 units in a GNG network for HRRs representing both simple and complex test structures for the transformation $x_1 \rightarrow x_2 \Longleftrightarrow \neg x_1 \vee x_2$.

these constituents in the expressions. These findings support our argument that the generalisation of HRR transformations is not based on the similarity of constituents in an expression, but on the position or role a constituent takes on. This is in stark contrast to holistic transformations of RAAM representations where generalisations to novel elements are based on the similarity of these elements to the elements used for training (cf. Chapter 3) and which have yet to achieve systematic processing of level 4 or higher. Whether such generalisations are possible based on the similarity of constituents in an expression is questionable, however, because they would require a strong similarity between structurally different expressions, namely the single elements in the training examples and more complex structures in the test examples. The results presented in Section 5.3 show, however, that it is exactly the structural difference between objects represented in RAAMs that determines the clustering of the representational space. Different classes of structures, i.e., structurally different objects, always fall into different clusters.

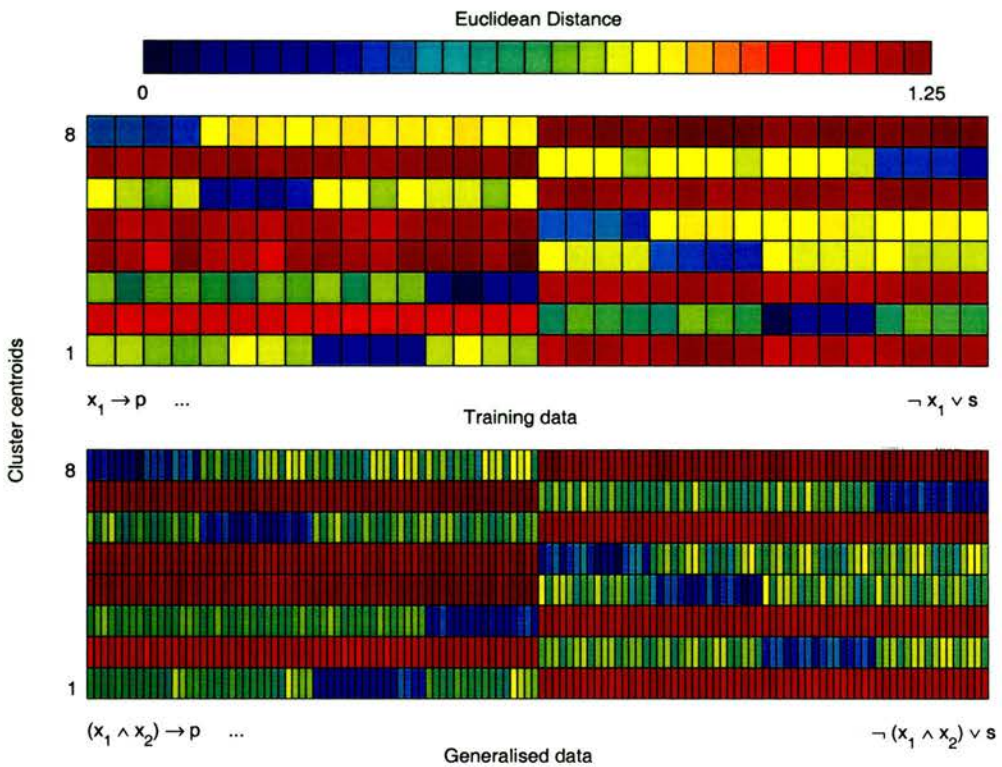

Figure 5.13: Distances between the cluster centroids and data points representing transformed training structures $x_1 \rightarrow x_2$ (top left) and $\neg x_1 \vee x_2$ (top right) and transformed complex structures $(x_1 \wedge x_2) \rightarrow x_3$ (bottom left) $\neg (x_1 \wedge x_2) \vee x_3$ (bottom right) with $x_1, x_2 \in \{p, q, r, s\}$. Test and training structures with the same instance of the rightmost variable fall into the same cluster.

5.5 Summary

Operations over hierarchical structured objects such as their mapping, classification, and transformation are structure-sensitive processes. The successful application of such processes to connectionist representations of hierarchical structures suggests that these representations must carry some form of structural information about the objects they represent. Symbolic representations possess a constituent structure whose syntactic manipulation provides the basis for structure-sensitive processing. Such constituent structure can usually not be found in connectionist representations, however. A different explanation is thus required for the successful application of structure-sensitive operations to connectionist representations of hierarchical structures.

The analysis of representations formed in the hidden layer of RAAMs contrasts Symbolists' assumption that connectionist representations in general, and connectionist representations of structured objects in particular, are completely unstructured. Rather, they possess a spatial structure, i.e., representations of similar objects are placed close together in the representational space, whereas representations of dissimilar objects are well separated. The experiments discussed here show that representations learned in the hidden layer of a RAAM form clusters of representations for structurally similar objects. Different classes of structures form clusters in different areas of the representational space and within each class of structures, clusters are formed with respect to the constituent last presented to the RAAM during training. In cases where this clustering complies with the distinction of objects required by the task a holistic operation implements, the holistic operation can be learned by a separately trained connectionist network. Alternatively, parallel learning of the encoding in the RAAM and a holistic operation alters the spatial structure of the representations as to provide the structural information necessary for the holistic operation. These observations led us to the conclusion that it is exactly this spatial structure of RAAM representations, i.e., the position of the representations in the representational space, that provides the information necessary for their structure-sensitive processing.

A systematic spatial structure of representations can also be observed for HRRs. No analysis of the representational space of HRRs had been carried out and presented in the literature so far. We applied a constructivist neural network to HRRs representing expressions in propositional logic which served as inputs for or were results of holistic transformations. Input structures and transformed structures cluster in the same principal way. Different classes of structures form clusters in different areas of the representational space. Within each class of structures, clusters can be formed with respect to different top-level constituents in the expressions.

Since the holistic transformation of HRR structures by circular convolution is a linear transformation, it is crucial for its successful application that the principle structure of the representational space within each cluster is the same for inputs and targets of the transformation. This is ensured by the similarity preserving properties of both the binding and the superposition operation. The structures $\mathbf{A_1}$ and $\mathbf{A_2}$ in the cluster representing the input to a transformation will have approximately the same spatial

relation as the corresponding structures $\mathbf{B_1}$ and $\mathbf{B_2}$ in the output cluster. In other words, the two clusters have approximately the same 'shape', allowing for a linear transformation in the representational space. As seen for the generalisation of transformations of RAAM structures in Figure 5.8, if the transformation process does not preserve the spatial relation of structures within a cluster during transformation, the transformed structures cannot be decoded correctly.

The more flexible clustering of HRRs in comparisons to RAAM structures can be explained by the more flexible way HRRs are constructed from elements in a domain. The number of constituents presented to the RAAM at a time, and hence the possible clustering of the RAAM structures, is restricted by the RAAM architecture. The process of constructing HRR structures, however, is not subject to such restrictions. Consequently, more constituents can have the same weight in the final representations than in the representations formed in RAAMs, where the one or two constituents last presented during encoding have a larger weight in the final representation than constituents presented earlier. As a result we would expect a wider range of holistic operations to be applicable to HRR structures without demanding the modification of the representational space towards the holistic operation at hand.

The results presented in this Chapter show that the systematic structure of the representational space of HRRs supports not only the acquisition of a transformation vector from examples but the generalisation of holistic transformations to structures containing novel elements and structures of higher complexity. Novel elements and more complex constituents are placed into the appropriate clusters according to their role or position within the structures. This behaviour could not be observed for RAAM structures.

The preservation of the principal spatial structure of HRRs during holistic transformations further suggests that a number of such transformations could be successively applied to the structures, or other holistic operations such as the classification of structures or their mapping could be applied to already transformed structures. The noise introduced by the transformation which manifests itself in smaller distances between the clusters might, however, require an occasional clean-up of intermediate results.

Chapter 6

Parallel Transformation of Different Classes of HRR Structures

We have seen in previous Chapters that holistic transformations of RAAM and HRR structures can be learned from examples. After training, the learning system was usually able to perform and generalise a transformation in two directions. When trained with structures of class A and class B as inputs and class B and class A as respective targets, the TN for RAAM structures and the learning algorithms for HRR structures were able to learn the transformation

$$\text{class A} \iff \text{class B}\,.$$

This means that the learning systems had in fact acquired two different, although related, transformations of the input structures. The same was observed for the construction of transformation vectors for HRR structures discussed in Section 3.3.2. Despite the limitations of this method, we were able to construct a transformation vector representing the transformation of a class of HRR structures and its reverse. These observations raise the question whether learning or constructing a transformation vector is also possible for a number of unrelated transformations at the same time. As discussed in Chapter 4 a truly useful transformational inference system based on holistic transformations of connectionist representations for logical expressions, for example, would require the acquisition of a number of different transformational inference rules and their appropriate application to structurally different input expressions.

Niklasson and Sharkey (1997) presented some data suggesting that a TN can learn to transform a number of different classes of RAAM structures in parallel. In this Chapter we investigate whether the representation of hierarchical objects by HRRs also allows for the parallel transformation of different classes of structures. Using the example of 13 different transformations, we analyse the problems that arise for their application in parallel, compare the constructed and learned transformation vectors with respect to their capacity for parallel transformations, and identify the conditions on the representations of different classes of structures that allow them to be transformed in parallel. Although we could not establish a general mechanism for the parallel transformation of different HRR structures within the scope of this thesis, the solutions found for the particular examples investigated here might provide a useful guideline for future research.

6.1 The Problem of Parallel Transformations

In comparison to the transformation of a single class of structures, performing a number of different transformations in parallel is a much more complex process. The system performing the task not only has to modify every structure according to a single transformation, it also has to choose which parts of all the transformational knowledge to apply to a particular input structure. In other words, it has to identify the appropriate changes that have to be made for every input structure separately. The only source of information available to the system is the input structure itself, however. The choice of modifications has thus to be made depending upon the elements a structure contains and upon their structural arrangement only.

Intuitively, we would expect problems in this process to arise, if the transformation network or the transformation vector has to transform classes of structures which are very similar, i.e., classes of objects that have a similar syntactic structure and contain the same elements. Such structures would be represented by vectors with very close positions in the representational space. Clusters comprised of representations for similar classes might therefore not be as clearly separated as is required for their transformation according to different transformation rules.

A second problem might arise from the noise introduced during a transformation.

Plate (1994, p150) observes that after transforming a HRR vector with a constructed transformation vector, the "strength of non-noise components in the transformed HRR is $1/\sqrt{k}$ times their strength in the original HRR, where k is the number of components in the transformation vector." As the number of components in the transformation vector increases, this factor becomes relatively small. As we will see in Section 6.1.2 the number of components in a constructed transformation vector increases significantly, when the vector has to perform a number of different transformations, which could cause the signal-to-noise ratio in the result structures to become too small for a correct decoding of the transformed elements. Learned transformations also introduce noise to the transformation results which was seen in the cluster analysis presented in Chapter 5. Clusters of transformed HRR structures, although still clearly separable, were generally closer in representational space than the corresponding clusters of inputs to the transformation, indicating that the result structures are noisier than the input structures. Again, the amount of noise might increase with the number of transformations performed by the learned transformation vectors.

For the investigation of these two potential problems we generated the following set of transformations in the domain of propositional logic:

$$(x_1 \rightarrow x_2) \wedge x_1 \implies x_2 \tag{6.1}$$

$$x_1 \wedge \neg x_2 \iff \neg (x_1 \rightarrow x_2) \tag{6.2}$$

$$\neg x_1 \vee \neg x_2 \iff \neg (x_1 \wedge x_2) \tag{6.3}$$

$$\neg x_1 \wedge \neg x_2 \iff \neg (x_1 \vee x_2) \tag{6.4}$$

$$x_1 \rightarrow x_2 \iff \neg x_1 \vee x_2 \tag{6.5}$$

$$x_1 \wedge x_2 \iff \neg (x_1 \rightarrow \neg x_2) \tag{6.6}$$

$$x_1 \vee x_2 \iff \neg (\neg x_1 \wedge \neg x_2) \tag{6.7}$$

The transformations were chosen such that they include classes of structures susceptible to both problems identified above. A number of transformations contain very similar classes of structures. For example, the structures on the right-hand side of transformations 6.2, 6.3, and 6.4 differ from the structures on the left-hand side of transformations 6.5, 6.6, and 6.7 in the negation of the whole expression only. The left-hand sides of transformations 6.2 and 6.4 only differ in the negation of x_1. If a single transformation vector is able to correctly transform these structures, it has to be

sensitive to these subtle differences. Further, the structures show different depth of embedding elements, reaching from a single element in transformation 6.1 to three levels of embedding on the right-hand side of transformations 6.6 and 6.7. These deeper embedded elements should be very sensitive to noise in the structures. Note that, where possible, transformations in both directions will be considered, which provides us with 13 different transformations to be performed in parallel.

6.1.1 Learning the Transformation Vector

Before exploring the acquisition of several transformations in parallel, we have to ensure that all transformations above can be learned separately. We applied the one-shot learning algorithm to all 13 transformations with 25 training examples formed using 5 different values for x_1 and x_2 and 25 test examples derived from a second set of 5 values for x_1 and x_2. After learning, the acquired transformation vector was tested on all training and test examples by decoding the elements of all output structures. For all 13 transformations the learned vector transformed both training and test examples with 100% accuracy.

We then took a number of different combinations of the transformations 6.1 to 6.4 to form single sets. Except for transformation 6.1, training was performed for transformations in both directions, requiring the transformation vector to implement up to 7 different transformations at the same time. After learning, the result structures for all training and test examples were again decoded into their elements. The results of 8 different tests are presented in Table 6.1.

As Table 6.1 reveals, errors occurred in all tests where the transformations 6.2 and 6.4 were combined. The further combination of these transformations with transformations 6.1 and 6.3 did not significantly change the number of errors made during decoding. The transformations 6.2 and 6.4 contain the classes

$$x_1 \wedge \neg x_2 \qquad \neg (x_1 \rightarrow x_2) \qquad \text{and}$$
$$\neg x_1 \wedge \neg x_2 \qquad \neg (x_1 \vee x_2),$$

respectively, which are indeed very similar. The left-hand sides of the transformations only differ in the negation of x_1, and the right-hand sides only differ in the two elements $\rightarrow$ and $\vee$.

Test	Transformations	Elements	Errors
1	6.1 $(x_1 \to x_2) \wedge x_1 \implies x_2$ 6.2 $x_1 \wedge \neg x_2 \iff \neg(x_1 \to x_2)$	700	0
2	6.1 $(x_1 \to x_2) \wedge x_1 \implies x_2$ 6.3 $\neg x_1 \vee \neg x_2 \iff \neg(x_1 \wedge x_2)$	700	0
3	6.1 $(x_1 \to x_2) \wedge x_1 \implies x_2$ 6.4 $\neg x_1 \wedge \neg x_2 \iff \neg(x_1 \vee x_2)$	700	0
4	6.2 $x_1 \wedge \neg x_2 \iff \neg(x_1 \to x_2)$ 6.3 $\neg x_1 \vee \neg x_2 \iff \neg(x_1 \wedge x_2)$	1200	0
5	6.1 $(x_1 \to x_2) \wedge x_1 \implies x_2$ 6.2 $x_1 \wedge \neg x_2 \iff \neg(x_1 \to x_2)$ 6.3 $\neg x_1 \vee \neg x_2 \iff \neg(x_1 \wedge x_2)$	1300	0
6	6.2 $x_1 \wedge \neg x_2 \iff \neg(x_1 \to x_2)$ 6.4 $\neg x_1 \wedge \neg x_2 \iff \neg(x_1 \vee x_2)$	1200	61 (5.08%)
7	6.1 $(x_1 \to x_2) \wedge x_1 \implies x_2$ 6.2 $x_1 \wedge \neg x_2 \iff \neg(x_1 \to x_2)$ 6.4 $\neg x_1 \wedge \neg x_2 \iff \neg(x_1 \vee x_2)$	1300	53 (4.07%)
8	6.1 $(x_1 \to x_2) \wedge x_1 \implies x_2$ 6.2 $x_1 \wedge \neg x_2 \iff \neg(x_1 \to x_2)$ 6.3 $\neg x_1 \vee \neg x_2 \iff \neg(x_1 \wedge x_2)$ 6.4 $\neg x_1 \wedge \neg x_2 \iff \neg(x_1 \vee x_2)$	1900	73 (3.84%)

Table 6.1: Results of combining the acquisition of up to 7 transformations in one transformation vector. The combined transformations, the number of elements in the output structures, and the number of wrongly decoded elements are shown.

Table 6.2 lists the places of decoding errors observed in tests 6, 7, and 8. Surprisingly, the majority of errors occurred when decoding the operations $\to$ and $\vee$. These errors were caused by the transformations from left to right:

$$x_1 \wedge \neg x_2 \implies \neg(x_1 \to x_2) \quad \text{and}$$
$$\neg x_1 \wedge \neg x_2 \implies \neg(x_1 \vee x_2)\,.$$

The reverse transformations were performed almost perfectly, however, although the

two input classes are structurally equivalent and only differ in the operations.

Test	Positions of Errors	Errors (in %)
6	$\rightarrow$ in $\neg(x_1 \rightarrow x_2)$	38
	$\vee$ in $\neg(x_1 \vee x_2)$	23
7	$\rightarrow$ in $\neg(x_1 \rightarrow x_2)$	15
	$\vee$ in $\neg(x_1 \vee x_2)$	34
	x_1 in $\neg(x_1 \vee x_2)$	4
8	$\rightarrow$ in $\neg(x_1 \rightarrow x_2)$	15
	$\vee$ in $\neg(x_1 \vee x_2)$	52
	x_1 in $\neg(x_1 \rightarrow x_2)$	4
	$\wedge$ in $\neg x_1 \wedge \neg x_2$	2

Table 6.2: Positions and number of errors when decoding the results of the parallel transformations in tests 6, 7, and 8 of Table 6.1.

A closer look at the way the representations for these four classes of structures are formed shows why the errors occurred in these particular positions. Let the elements *operation, negation, implication, disjunction, antecedent, consequent, first disjunction operand, second disjunction operand, x,* and y be represented by a set of randomly chosen HRR vectors {**op, neg, impl, disj, ante, cons, d_first, d_second, X, Y**}. Then representations for the structures $\neg(x_1 \rightarrow x_2)$ and $\neg(x_1 \vee x_2)$ are formed as follows with $x_1 = x$ and $x_2 = y$:

$$\textbf{not\_(X\_impl\_Y)} = \textbf{not} \circledast (\textbf{op} \circledast \textbf{impl} + \textbf{ante} \circledast \textbf{X} + \textbf{cons} \circledast \textbf{Y})$$

$$\textbf{not\_(X\_or\_Y)} = \textbf{not} \circledast (\textbf{op} \circledast \textbf{disj} + \textbf{d\_first} \circledast \textbf{X} + \textbf{d\_second} \circledast \textbf{Y}) .$$

Let further the elements *conjunction, first conjunction operand,* and *second conjunction operand* be represented by a set of randomly chosen HRR vectors {**conj, first, second**}. Then representations for the structures $\neg x_1 \wedge \neg x_2$ and $x_1 \wedge \neg x_2$ are formed as follows with $x_1 = x$ and $x_2 = y$:

$$\textbf{not\_X\_and\_not\_Y} = \textbf{op} \circledast \textbf{conj} + \textbf{first} \circledast \textbf{not} \circledast \textbf{X} + \textbf{second} \circledast \textbf{not} \circledast \textbf{Y}$$

$$\textbf{X\_and\_not\_Y} = \textbf{op} \circledast \textbf{conj} + \textbf{first} \circledast \textbf{X} + \textbf{second} \circledast \textbf{not} \circledast \textbf{Y} .$$

Although the first two expressions are structurally equivalent, their HRR representations differ in a number of elements, namely in the two fillers convolved with **op** and in the roles **ante** and **d_first** and **cons** and **d_second**, respectively. The HRR representations for the structures $\neg x_1 \wedge \neg x_2$ and $x_1 \wedge \neg x_2$, however, differ only in the presence and absence of the negation in the second constituent. Recall from Chapter 2 that circular convolution is randomising. We can therefore expect the two vectors **first** $\circledast$ **not** $\circledast$ **X** and **first** $\circledast$ **X** to be different. However, the superposition of HRR vectors preserves similarity. The superpositions of these two vectors with both the vectors **op** $\circledast$ **conj** and **second** $\circledast$ **not** $\circledast$ **Y** will therefore be similar to each other.

Whereas the apparent similarity of the classes $\neg (x_1 \rightarrow x_2)$ and $\neg (x_1 \vee x_2)$ is not reflected in their HRRs (all superimposed bindings are different), the similarity of the classes $\neg x_1 \wedge \neg x_2$ and $x_1 \wedge \neg x_2$ is preserved in their respective HRR vectors. The strong similarity of these HRRs might have caused the errors in their parallel transformation.

The similarity of HRR vectors is reflected in their spatial structure, i.e., in their position in the representational space. In the cluster analyses presented in the previous Chapter the representational spaces of HRR structures were generally clustered such that vectors representing members of the same class of structures fell into the same cluster and each cluster for a class of structures was well separated from clusters representing different classes. We argued that it is this clear spatial structure of HRRs that provides the information necessary for their holistic transformation and allows a single transformation vector to perform both a transformation and its reverse. However, all classes of structures considered in Chapter 5 differed in a number of elements. The difference of only a single element in the representations of the classes $\neg x_1 \wedge \neg x_2$ and $x_1 \wedge \neg x_2$ on the other hand might result in a representational space where vectors representing members of the two different classes are not as clearly separated. This representational space might hence not provide enough information to distinguish the conjunction in $x_1 \wedge \neg x_2$ that has to be transformed into an implication by transformation 6.2 from the conjunction in $\neg x_1 \wedge \neg x_2$ to be transformed into a disjunction by transformation 6.4.

In order to test this hypothesis, we performed a cluster analysis of the training examples used in test 8 for learning the transformations 6.1 to 6.4 in parallel. Training

was performed for all possible structures with $x_1, x_2 \in \{p, q, r, s, t\}$. From the considerations above, we expected the two clusters representing the classes $\neg x_1 \wedge \neg x_2$ and $x_1 \wedge \neg x_2$ not to be as clearly separated as clusters formed for representations of all other classes.

Cluster	Cluster members			
1	$(x_1 \rightarrow x_2) \wedge x_1$			
2	$\neg (x_1 \rightarrow x_2)$			
3	$\neg x_1 \vee \neg x_2$			
4	$\neg p \wedge \neg p$	$\neg p \wedge \neg q$	$\neg p \wedge \neg s$	
	$\neg q \wedge \neg p$	$\neg q \wedge \neg q$	$\neg q \wedge \neg s$	
	$\neg r \wedge \neg p$	$\neg r \wedge \neg q$	$\neg r \wedge \neg s$	
	$\neg s \wedge \neg p$	$\neg s \wedge \neg q$	$\neg s \wedge \neg s$	
	$\neg t \wedge \neg p$	$\neg t \wedge \neg q$	$\neg t \wedge \neg s$	$\neg t \wedge \neg t$
	$p \wedge \neg p$	$p \wedge \neg q$	$p \wedge \neg s$	$p \wedge \neg t$
	$q \wedge \neg p$	$q \wedge \neg q$	$q \wedge \neg s$	
	$r \wedge \neg p$	$r \wedge \neg q$	$r \wedge \neg s$	
	$s \wedge \neg p$	$s \wedge \neg q$	$s \wedge \neg s$	
	$t \wedge \neg p$	$t \wedge \neg q$	$t \wedge \neg s$	
5	$\neg (x_1 \wedge x_2)$			
6	$\neg (x_1 \vee x_2)$			
7		$\neg p \wedge \neg r$	$\neg p \wedge \neg t$	
		$\neg q \wedge \neg r$	$\neg q \wedge \neg t$	
		$\neg r \wedge \neg r$	$\neg r \wedge \neg t$	
		$\neg s \wedge \neg r$	$\neg s \wedge \neg t$	
		$\neg t \wedge \neg r$		
		$p \wedge \neg r$		
		$q \wedge \neg r$	$q \wedge \neg t$	
		$r \wedge \neg r$	$r \wedge \neg t$	
		$s \wedge \neg r$	$s \wedge \neg t$	
		$t \wedge \neg r$	$t \wedge \neg t$	

Table 6.3: Clusters formed after constructing a layer of 7 units in a GNG network for HRRs representing the structures used for learning the transformations 6.1 to 6.4 in parallel.

A GNG network was trained to cluster the representational space into 7 clusters, in

accordance with the number of different input classes of structures present in the seven transformations. New units were added to the network after every 100 cycles through all training examples. The results of the clustering are reported in Table 6.3.

Most clusters contain all structures belonging to exactly one class. However, clusters 4 and 7 both contain structures belonging to the classes $\neg x_1 \wedge \neg x_2$ and $x_1 \wedge \neg x_2$. With two exceptions, structures of both classes with the constituents $\neg p$, $\neg q$, and $\neg s$ as second operand fall into cluster 4, whereas structures of both classes with the constituents $\neg r$ and $\neg t$ as second operand form cluster 7. As expected, the two classes are not clearly separated. Figure 6.1 shows the distances between the cluster centroids

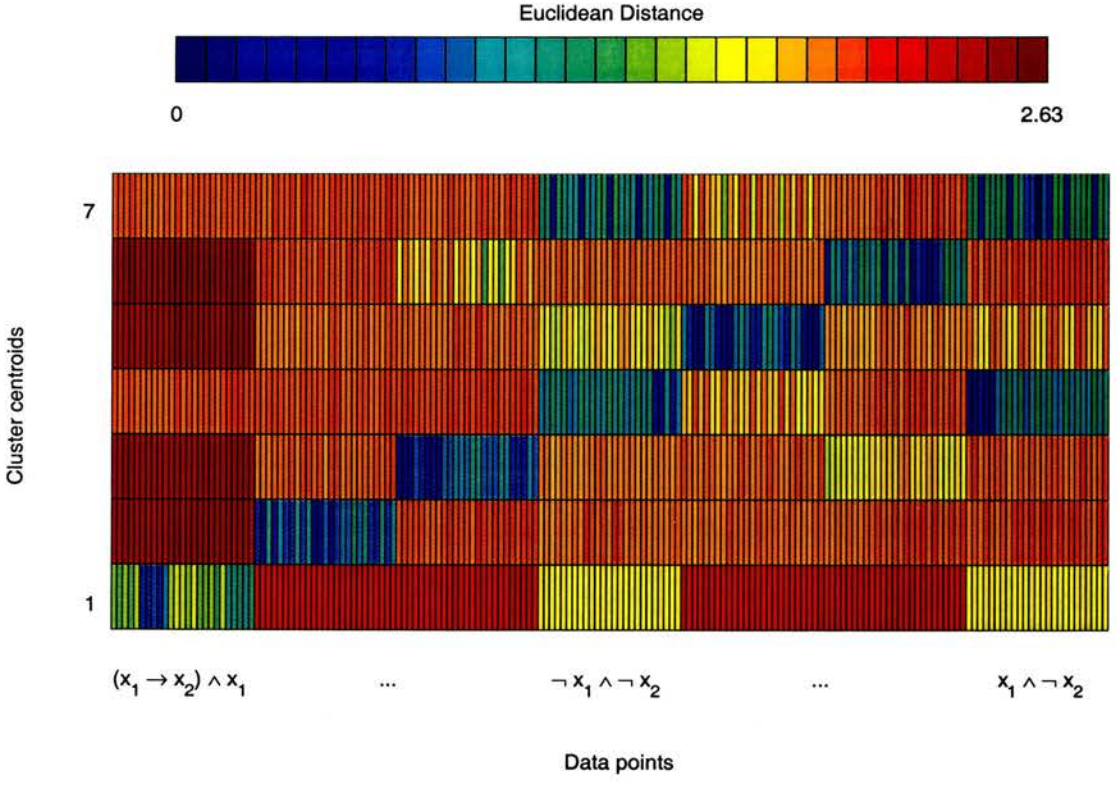

Figure 6.1: Distances between the cluster centroids and data points representing the HRR structures used for learning the transformations 6.1 to 6.4 in parallel.

and all data points. The data points are ordered along the x-axis as presented in Table 6.3 with $\neg x_1 \wedge \neg x_2$ as fourth and $x_1 \wedge \neg x_2$ as seventh class. The colour coding illustrates again that the members of these two classes are close to the centroids of both cluster 4 and cluster 7. This similar spatial structure of the classes $\neg x_1 \wedge \neg x_2$

and $x_1 \wedge \neg x_2$ is likely to have caused the problems for the transformation of the two classes in parallel.

6.1.2 Constructing the Transformation Vector

Another method of holistically transforming HRR structures is to construct a transformation vector as proposed by Plate (1994) and discussed in Section 3.3.2. When first introducing this technique, Plate already anticipated the construction of a transformation vector for a transformation in both directions. For example, the structure

$$\mathbf{S_I} = \mathbf{r}_1 \circledast \mathbf{X} + \mathbf{r}_2 \circledast \mathbf{Y}$$

can be transformed into the structure

$$\mathbf{S_O} = \mathbf{r}_1 \circledast \mathbf{A} + \mathbf{r}_2 \circledast \mathbf{B}$$

by a transformation vector

$$\mathbf{T_{IO}} = \mathbf{X}' \circledast \mathbf{A} + \mathbf{Y}' \circledast \mathbf{B}$$

which can be interpreted as 'replace the filler X by the filler A and the filler Y by the filler B.' The reverse transformation is achieved by the transformation vector

$$\mathbf{T_{OI}} = \mathbf{A}' \circledast \mathbf{X} + \mathbf{B}' \circledast \mathbf{Y}\,.$$

Both transformation vectors can now be superimposed to a vector

$$\mathbf{T} = \mathbf{T_{IO}} + \mathbf{T_{OI}}$$

which is able to transform the structure $\mathbf{S_I}$ into $\mathbf{S_O}$ and the structure $\mathbf{S_O}$ into $\mathbf{S_I}$. Plate (1994) observed that the superposition of transformation vectors increases the amount of noise introduced during the transformation. It does not make the transformed structures unrecognisable, however, if the vector dimension is high enough. Our results presented in Section 3.3.2 support this observation. However, applying this method to the transformations 6.2 and 6.4 turns out to be as problematic as learning the transformation vector from examples. The reasons for the problems become apparent in the following analysis.

To keep things simple we will only consider the transformations 6.2 and 6.4 from left to right, which is the direction of the transformations that caused problems when the transformation vector was learned. A vector performing the transformation

$$x_1 \wedge \neg x_2 \implies \neg (x_1 \rightarrow x_2)$$

can be constructed as

$$\mathbf{T}_1 = \mathbf{not} \circledast (\mathbf{conj}' \circledast \mathbf{impl} + \mathbf{first}' \circledast \mathbf{ante} + \mathbf{second}' \circledast \mathbf{not}' \circledast \mathbf{cons}) .$$

This transformation vector encodes the following four changes necessary for the correct transform:

1. The conjunction is replaced by an implication.
2. The first operand of the conjunction becomes the antecedent of the implication.
3. The non-negated second operand of the conjunction becomes the consequent of the implication.
4. The whole resulting structure is negated.

Using the distributivity of convolution over vector addition, $\mathbf{T}_1$ can be reduced to

$$\mathbf{T}_2 = \mathbf{not} \circledast \mathbf{conj}' \circledast \mathbf{impl} + \mathbf{not} \circledast \mathbf{first}' \circledast \mathbf{ante} + \mathbf{second}' \circledast \mathbf{cons} .$$

Note that **not**$'$ is only the approximate inverse of **not**, and therefore $\mathbf{T}_1 \approx \mathbf{T}_2$. However, we generally found that such a reduced vector transformed the given structures more accurately than the originally constructed one. This can be explained by the fact that reducing the number of changes to a structure which are encoded in the transformation vector also reduces the amount of noise introduced by the transformation. For example, when applying the transformation vector $\mathbf{T}_1$ to $x_1 \wedge \neg x_2$, the negation of x_2 is 'deleted', but re-introduced with the negation of the whole structure in the same transformation. These two operations cause more noise in the result structure than leaving the negation of x_2 unchanged as is the case when applying $\mathbf{T}_2$. It is therefore preferred to use the transformation vector encoding the smallest number of changes to a structure. We applied $\mathbf{T}_2$ to 100 structures of the class $x_1 \wedge \neg x_2$. All elements of all structures were correctly transformed and decoded.

A similar transformation vector can be constructed for transformation 6.4 from left to right:

$$\neg x_1 \wedge \neg x_2 \implies \neg (x_1 \vee x_2) .$$

$$\mathbf{T}_3 = \mathbf{not} \circledast (\mathbf{conj}' \circledast \mathbf{disj} + \mathbf{first}' \circledast \mathbf{not}' \circledast \mathbf{d\_first} + \mathbf{second}' \circledast \mathbf{not}' \circledast \mathbf{d\_second})$$

which can be simplified to

$$\mathbf{T}_4 = \mathbf{not} \circledast \mathbf{conj}' \circledast \mathbf{disj} + \mathbf{first}' \circledast \mathbf{d\_first} + \mathbf{second}' \circledast \mathbf{d\_second} .$$

Again, when $\mathbf{T}_4$ was tested on 100 structures of the class $\neg x_1 \wedge \neg x_2$, all elements were correctly transformed and decoded. In order to perform both transformations with a single transformation vector we superimpose $\mathbf{T}_2$ and $\mathbf{T}_4$ to the vector

$$\begin{aligned}\mathbf{T}_{2+4} = {} & \mathbf{not} \circledast \mathbf{conj}' \circledast \mathbf{impl} + \mathbf{not} \circledast \mathbf{first}' \circledast \mathbf{ante} + \mathbf{second}' \circledast \mathbf{cons} \\ & + \mathbf{not} \circledast \mathbf{conj}' \circledast \mathbf{disj} + \mathbf{first}' \circledast \mathbf{disj\_first} + \mathbf{second}' \circledast \mathbf{disj\_second} .\end{aligned}$$

Using this transformation vector for either transformation 6.2 or transformation 6.4 is problematic, however. In order to keep the notation relatively short we first look at the parts of the vectors regarding the two operations only[1]. As shown in the previous Section, instances of both classes $\neg x_1 \wedge \neg x_2$ and $x_1 \wedge \neg x_2$ are represented by HRRs of the form $(\mathbf{op} \circledast \mathbf{conj} + \ldots)$. Thus, convolving $\mathbf{T}_{2+4}$ with an instance of either class leads to

$$(\mathbf{op} \circledast \mathbf{conj} + \ldots) \circledast \mathbf{T}_{2+4} =$$

$$(\mathbf{op} \circledast \mathbf{conj} + \ldots) \circledast (\mathbf{not} \circledast \mathbf{conj}' \circledast \mathbf{impl} + \mathbf{not} \circledast \mathbf{conj}' \circledast \mathbf{disj} + \ldots) \approx$$

$$(\mathbf{op} \circledast \mathbf{not} \circledast \mathbf{impl} + \mathbf{op} \circledast \mathbf{not} \circledast \mathbf{disj} + \ldots) \approx$$

$$\mathbf{not} \circledast (\mathbf{op} \circledast \mathbf{impl} + \mathbf{op} \circledast \mathbf{disj} + \ldots) + \mathit{noise} .$$

The structure resulting from this transformation is comprised of 18 terms which represent all combinations of the three constituents of the input structure with the six bindings the transformation vector consists of. Most of these terms, for example the

[1]Recall that when the transformation vector was learned from examples, the majority of decoding errors was observed for the elements *implication* and *disjunction* resulting from the transformation of a conjunction.

terms **op** ⊛ **conj** ⊛ **not** ⊛ **first′** ⊛ **ante** and **second** ⊛ **not** ⊛ **Y** ⊛ **first′** ⊛ **disj_first**, do not represent meaningful constituents of a structure and thus can be treated as extra noise. Plate (1994) refers to such terms as *cross terms*. However, the terms **op** ⊛ **impl** and **op** ⊛ **disj** are both valid parts of structures in the domain. Convolving them with the approximate or exact inverse of the role **op** results in the fillers **impl** and **disj**, respectively, which are both elements in the domain. Consequently, retrieving the filler convolved with **op** from a vector of the form **not** ⊛ (**op** ⊛ **impl** + **op** ⊛ **disj** + ...) will result in noisy versions of the elements **impl** and **disj** with equal probability. The result of the transformation thus can potentially represent both a structure of the class $\neg(\ldots \rightarrow \ldots)$ and a structure of the class $\neg(\ldots \vee \ldots)$.

Superimposing $\mathbf{T}_2$ and $\mathbf{T}_4$ in order to perform both transformation 6.2 and 6.4 from left to right vector causes an ambiguity in the transformed structures which we would expect to result in errors when decoding the fillers of the role *operation*. Using a superimposed constructed vector for the transformations in both directions is only different in that the number of cross terms, i.e., the amount of noise in the result structures, increases further. The ambiguity regarding the operation of the result structure remains[2].

We repeated the tests 6,7, and 8 (cf. Table 6.1) which include the transformations 6.2 and 6.4 now using constructed transformation vectors. The results are presented in Table 6.4 and Table 6.5. Similar to the results obtained from a learned transformation vector and as expected from the analysis above, most errors occurred when decoding the operations $\rightarrow$ and $\vee$ of structures transformed from the classes $\neg x_1 \wedge \neg x_2$ and $x_1 \wedge \neg x_2$. The small number of remaining errors were distributed over all other elements in the structures.

Interestingly, the number of errors was generally about twice as high for the constructed transformation vector as for the learned one. Further tests of the construction method including other combinations of the transformations 6.1 to 6.4 revealed that some errors also occurred for constructed transformations which showed 100% decod-

[2]Looking at the operands of the result structures shows that applying $\mathbf{T}_{2+4}$ to a structure of the class $x_1 \wedge \neg x_2$ causes a further ambiguity, because the result structure contains the term **not** ⊛ **cons** ⊛ **Y** as well as **not** ⊛ **d_second** ⊛ **Y**. This ambiguity is hidden in the usual decoding process for elements, however, because it only becomes apparent, if the roles of a structure are decoded using the inverse of a filler as key.

ing accuracy when the learned transformation vector was used.

Test	Transformations			Elements	Errors
6	6.2	$x_1 \wedge \neg x_2$	$\Longleftrightarrow \neg(x_1 \rightarrow x_2)$		
	6.4	$\neg x_1 \wedge \neg x_2$	$\Longleftrightarrow \neg(x_1 \vee x_2)$	1200	152 (12.67%)
7	6.1	$(x_1 \rightarrow x_2) \wedge x_1$	$\Longrightarrow x_2$		
	6.2	$x_1 \wedge \neg x_2$	$\Longleftrightarrow \neg(x_1 \rightarrow x_2)$	1300	112 (8.62%)
	6.4	$\neg x_1 \wedge \neg x_2$	$\Longleftrightarrow \neg(x_1 \vee x_2)$		
8	6.1	$(x_1 \rightarrow x_2) \wedge x_1$	$\Longrightarrow x_2$		
	6.2	$x_1 \wedge \neg x_2$	$\Longleftrightarrow \neg(x_1 \rightarrow x_2)$	1900	140 (7.37%)
	6.3	$\neg x_1 \vee \neg x_2$	$\Longleftrightarrow \neg(x_1 \wedge x_2)$		
	6.4	$\neg x_1 \wedge \neg x_2$	$\Longleftrightarrow \neg(x_1 \vee x_2)$		

Table 6.4: Results of combining the construction of up to 7 different transformations in one transformation vector. The combined transformations, the number of elements in the output structures, and the number of wrongly decoded elements are shown.

Test	Positions of Errors			Errors (in %)
6	$\rightarrow$	in	$\neg(x_1 \rightarrow x_2)$	70
	$\vee$	in	$\neg(x_1 \vee x_2)$	59
	x_1	in	$\neg(x_1 \vee x_2)$	7
7	$\rightarrow$	in	$\neg(x_1 \rightarrow x_2)$	53
	$\vee$	in	$\neg(x_1 \vee x_2)$	33
	x_1	in	$\neg(x_1 \rightarrow x_2)$	11
	x_1	in	$\neg(x_1 \vee x_2)$	5
8	$\rightarrow$	in	$\neg(x_1 \rightarrow x_2)$	26
	$\vee$	in	$\neg(x_1 \vee x_2)$	72
	x_1	in	$\neg(x_1 \rightarrow x_2)$	7
	x_2	in	$\neg(x_1 \rightarrow x_2)$	9
	$\wedge$	in	$\neg x_1 \wedge \neg x_2$	7

Table 6.5: Positions and number of errors when decoding the results for the constructed parallel transformations of Table 6.4.

3 decoding errors occurred when the constructed transformation 6.3 was performed on its own. The combination of transformations 6.2 and 6.3 (cf. test 4 in Table 6.1 for the learned transformation) resulted in 17 decoding errors distributed over all elements in the structures. Although these error numbers are relatively small, they show that the construction method is not as reliable as the learning method. Since the errors in these transformations were not concentrated on particular elements but distributed over all constituents in the structures, we would expect the reason for them to be the noise introduced by the constructed transformation vector.

In comparison, when the transformation vectors were learned from examples, decoding errors for structures other than $\neg(x_1 \rightarrow x_2)$ and $\neg(x_1 \vee x_2)$ were only observed in test 8 which combined the largest number of parallel transformations, and they only involved 2 out 1300 elements (cf. Table 6.2). The amount of noise introduced by the learned transformation vectors does not seem to be so high. For the particular examples investigated so far, learning the transformation vector from examples is to be preferred over the construction technique, in terms of noise tolerance.

6.2 A Solution

For both methods, constructing a transformation vector and learning it from examples, the errors in the parallel transformation of structures were caused by the strong similarity of the HRR vectors representing the two classes of structures $x_1 \wedge \neg x_2$ and $\neg x_1 \wedge \neg x_2$. These representations were very close in representational space which did not permit the acquisition of a correct transformation vector from examples. Moreover, they contained identical role-filler bindings which could not be transformed unambiguously by a constructed transformation vector. A way to overcome these problems is to modify the encoding scheme for the structures such that representations for these two classes of structures, and in fact *all* classes in the domain, are sufficiently different. Since the convolution operation is randomising, the convolution of two similar structures with two different vectors should result in two new structures better separated than the original ones. We therefore introduced a *type* or *name* for every class of structures. In fact, Plate (1994) often used a similar mechanism which he called

frame name in his originally proposed encoding scheme[3]. For expressions in propositional logic the elements representing operations like *implication, disjunction,* and *conjunction* were not convolved with a role like *operation,* but served as the frame name which was superimposed with the remaining parts of the expression. An example of this encoding scheme can be seen in Section 3.3.2. However, this method makes it difficult to decode the operation of an expression and does not allow for different classes of structures with the same top-level operation in one domain. Moreover, the superposition of two otherwise similar structures with different frame names does not separate the structures as much as the convolution with frame names or types would do. We therefore kept the operation of a structure as a filler bound to the role **op** and introduced an additional class type. Our domain consisted of 13 different classes of structures. We therefore introduced a set of 13 new HRR vectors {**type_1, type_2** ... **type_13**}. Each structure of a class was then convolved with the appropriate type. For example, instances of the classes $\neg x_1 \wedge \neg x_2$ and $x_1 \wedge \neg x_2$ with $x_1 = x$ and $x_2 = y$ were now constructed as

$$\begin{aligned}
\textbf{not_X_and_not_Y} &= \textbf{type_1} \circledast (\,\textbf{op} \circledast \textbf{conj} + \textbf{first} \circledast \textbf{not} \circledast \textbf{X} + \textbf{second} \circledast \textbf{not} \circledast \textbf{Y}\,) \\
\textbf{X_and_not_Y} &= \textbf{type_2} \circledast (\,\textbf{op} \circledast \textbf{conj} + \textbf{first} \circledast \textbf{X} \quad + \textbf{second} \circledast \textbf{not} \circledast \textbf{Y}\,).
\end{aligned}$$

As a result, structures of these two classes no longer only differed in a single element but in all constituents they were composed of:

$$\begin{aligned}
\textbf{type_1} \circledast \textbf{op} \circledast \textbf{conj} &\neq \textbf{type_2} \circledast \textbf{op} \circledast \textbf{conj} \\
\textbf{type_1} \circledast \textbf{first} \circledast \textbf{not} \circledast \textbf{X} &\neq \textbf{type_2} \circledast \textbf{first} \circledast \textbf{X} \\
\textbf{type_1} \circledast \textbf{second} \circledast \textbf{not} \circledast \textbf{Y} &\neq \textbf{type_2} \circledast \textbf{second} \circledast \textbf{not} \circledast \textbf{Y}\,.
\end{aligned}$$

The HRRs for the structures $\neg\,(x_1 \rightarrow x_2)$ and $\neg\,(x_1 \vee x_2)$ with $x_1 = x$ and $x_2 = y$ were now constructed as follows:

$$\begin{aligned}
\textbf{not_(X_or_Y)} &= \textbf{type_3} \circledast \textbf{not} \circledast (\,\textbf{op} \circledast \textbf{disj} \;+ \textbf{d_first} \circledast \textbf{X} + \textbf{d_second} \circledast \textbf{Y}\,) \\
\textbf{not_(X_impl_Y)} &= \textbf{type_4} \circledast \textbf{not} \circledast (\,\textbf{op} \circledast \textbf{impl} + \textbf{ante} \circledast \textbf{X} \quad + \textbf{cons} \circledast \textbf{Y}\,)\;.
\end{aligned}$$

[3] A frame in Plate's notation represents what we call a class of structures. Two representations with the same frame share all roles and their structural arrangement but differ in the fillers.

6.2.1 Learning the Transformation Vector

We repeated the tests presented in Section 6.1.1 for learning the transformation vector now using the new encoding scheme. Again, the one-shot learning algorithm was first applied separately to all 13 transformations with 25 training examples and 25 different test examples. For all single transformations the learned vector transformed both training and test examples with 100% accuracy. We then combined the different transformations as before in Table 6.1. The results are presented in Table 6.6.

Test	Transformations	Elements	Errors
1	6.1 $(x_1 \rightarrow x_2) \wedge x_1 \implies x_2$ 6.2 $x_1 \wedge \neg x_2 \iff \neg(x_1 \rightarrow x_2)$	700	0
2	6.1 $(x_1 \rightarrow x_2) \wedge x_1 \implies x_2$ 6.3 $\neg x_1 \vee \neg x_2 \iff \neg(x_1 \wedge x_2)$	700	0
3	6.1 $(x_1 \rightarrow x_2) \wedge x_1 \implies x_2$ 6.4 $\neg x_1 \wedge \neg x_2 \iff \neg(x_1 \vee x_2)$	700	0
4	6.2 $x_1 \wedge \neg x_2 \iff \neg(x_1 \rightarrow x_2)$ 6.3 $\neg x_1 \vee \neg x_2 \iff \neg(x_1 \wedge x_2)$	1200	0
5	6.1 $(x_1 \rightarrow x_2) \wedge x_1 \implies x_2$ 6.2 $x_1 \wedge \neg x_2 \iff \neg(x_1 \rightarrow x_2)$ 6.3 $\neg x_1 \vee \neg x_2 \iff \neg(x_1 \wedge x_2)$	1300	19 (1.46%)
6	6.2 $x_1 \wedge \neg x_2 \iff \neg(x_1 \rightarrow x_2)$ 6.4 $\neg x_1 \wedge \neg x_2 \iff \neg(x_1 \vee x_2)$	1200	3 (0.25%)
7	6.1 $(x_1 \rightarrow x_2) \wedge x_1 \implies x_2$ 6.2 $x_1 \wedge \neg x_2 \iff \neg(x_1 \rightarrow x_2)$ 6.4 $\neg x_1 \wedge \neg x_2 \iff \neg(x_1 \vee x_2)$	1300	23 (1.77%)
8	6.1 $(x_1 \rightarrow x_2) \wedge x_1 \implies x_2$ 6.2 $x_1 \wedge \neg x_2 \iff \neg(x_1 \rightarrow x_2)$ 6.3 $\neg x_1 \vee \neg x_2 \iff \neg(x_1 \wedge x_2)$ 6.4 $\neg x_1 \wedge \neg x_2 \iff \neg(x_1 \vee x_2)$	1900	38 (2.00%)

Table 6.6: Results of combining the acquisition of up to 7 transformations of structures containing class types. The combined transformations, the number of elements in the output structures, and the number of wrongly decoded elements are shown.

With the exception of three decoding errors in test 6, for all tests of up to four parallel transformations, the transformation vector was learned correctly. Errors occurred when more than four transformations were learned in a single step. In contrast to the results for structures without class types, however, these errors were distributed over all elements in the structures and not restricted to the operations. This suggests that the errors were no longer caused by the similarity of input structures but by the noise produced during transformation and, in particular, during decoding. Using class types added a new level of embedding to every structure. As we have seen in Section 2.4, the number of elements that can be reliably decoded from a structure representation decreases with the embedding level of higher-order bindings in a structure. Increasing this level causes the signal-to-noise ratio in the recovered noisy elements to become smaller which in turn can result in decoding errors.

The cluster analysis of the input structures for test 8 shows the difference between the spatial structure of representations with and without types. The clusters detected by the GNG network and the distances between the cluster centroids and the data points are presented in Table 6.7 and Figure 6.2, respectively. The GNG network formed

Cluster	Cluster members
1	$(x_1 \rightarrow x_2) \wedge x_1$
2	$\neg\,(x_1 \rightarrow x_2)$
3	$\neg\, x_1 \vee \neg\, x_2$
4	$\neg\, x_1 \wedge \neg\, x_2$
5	$\neg\,(x_1 \wedge x_2)$
6	$\neg\,(x_1 \vee x_2)$
7	$x_1 \wedge \neg\, x_2$

Table 6.7: Clusters formed after constructing a layer of 7 units in a GNG network for HRRs representing structures with class types used for learning the transformations 6.1 to 6.4 in parallel.

7 clusters, each containing structures of exactly one class. Cluster 4 and cluster 7 now consisted of all structures belonging to the classes $\neg\, x_1 \wedge \neg\, x_2$ and $x_1 \wedge \neg\, x_2$, respectively. The two classes were clearly separated in the representational space. A comparison of Figure 6.2 with Figure 6.1 further shows that the representational

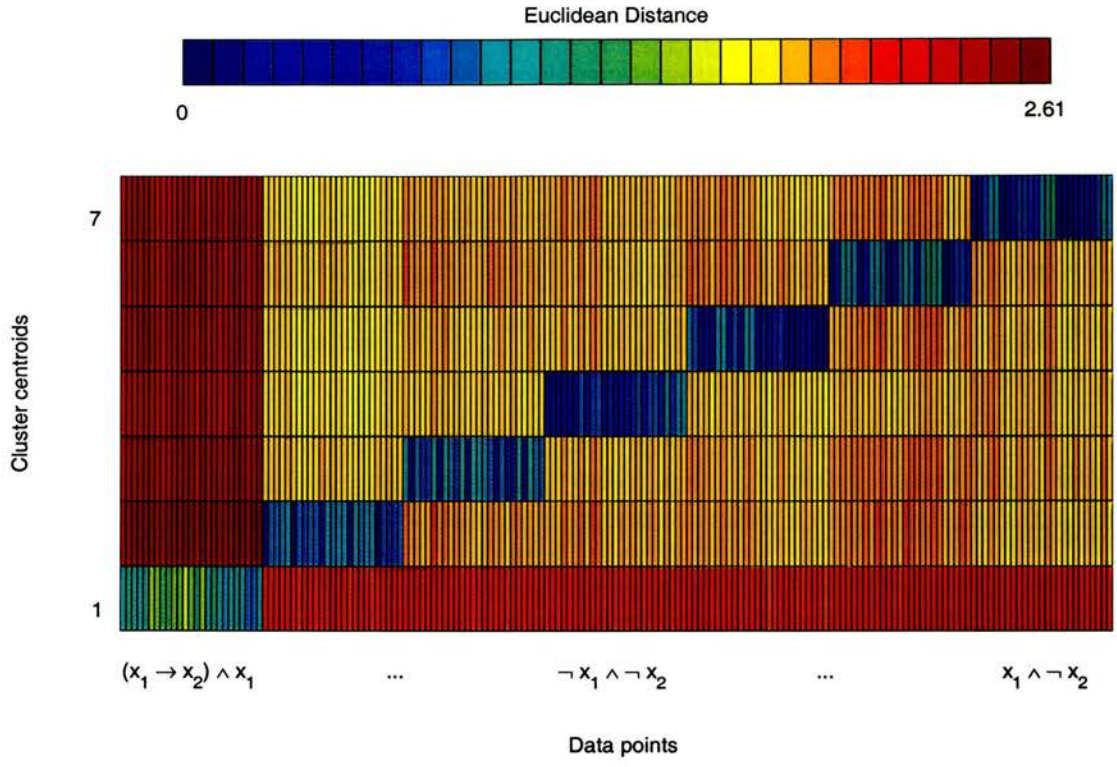

Figure 6.2: Distances between the cluster centroids and data points representing HRR structures with class types used for learning the transformations 6.1 to 6.4 in parallel.

space now had a better overall structure. Whereas in Figure 6.1 some members of some classes are relatively close to not only their own cluster centroid but also to one or two others, the distances of the data points to their own cluster centroids are now consistently smaller than their distances to other cluster centroids[4]. The use of class types results in a clearer spatial structure of the representations. This supports our argument that the errors observed in tests with more than four transformations were not caused by the similarity of representations but by the noise introduced during transformation and decoding.

This noise can be dealt with relatively easily, however, by increasing the dimensions of the HRR vectors and the number of structures in the training set. So far 4096-dimensional HRR vectors were used to represent elements and structures. We

[4]The larger distance of cluster 1 to all other clusters can be explained by the fact that cluster 1 contains the class $(x_1 \rightarrow x_2) \wedge x_1$ which has a structural arrangement of the constituents different from all other classes. It is thus even less similar to other classes than all other classes amongst themselves.

increased the size of the vectors to 8192. This particular size was chosen to make the best use of the implementation of circular convolution by Fast Fourier Transforms. When test 8 was repeated with the larger HRR vectors, the number of errors was reduced to 9 (0.47%). Increasing the number of structures in the training set by adding more values for x_1 and x_2 further reduced the number of errors. When the transformation vector was learned from all structures containing 7 different values for x_1 and x_2 and tested on structures with 10 different values, only one decoding error was observed out of possible 1900. The extension of the test set to 12 different values for x_1 and x_2 resulted in only two decoding errors out of 2736 elements in the structures. Increasing the vector dimension and the size of the training set thus reduced the decoding errors to less than 0.1%. The learned transformation vector very accurately performed the 7 transformations of structures with class types in parallel.

Finally we combined transformations 6.1 to 6.4 with the remaining transformations introduced at the beginning of the Chapter. Note that all three remaining transformations contain structures that differed in our original encoding scheme from structures in previously added transformations by only a single constituent. Training was performed for 7 different values of x_1 and x_2. The test set contained all structures obtained from 10 different values for the two variables. 8192-dimensional HRR vectors were used to encode the structures. Combining transformations 6.1 to 6.4 with transformations 6.6 and 6.7 provided a test set of structures containing 3100 elements. All but 10 elements (0.32%) were correctly transformed and decoded. The acquisition of all 13 transformations in parallel provided a test set of structures containing 3700 elements. Only 15 (0.41%) of these elements were wrongly decoded. In both cases the observed errors were distributed over various elements in the structures.

6.2.2 Constructing the Transformation Vector

The changes in our encoding scheme and the increase of the vector dimensionality enabled the learning algorithm to learn the parallel transformation of 13 classes of structures with a very high accuracy. Since the learning algorithm is efficient, easy to apply to a number of structures and, as we have seen above, reliable, it makes the construction of a transformation vector unnecessary, in particular since the construction process becomes more complicated the more transformations are considered. More-

over, the results presented in Section 6.1.2 indicated that the construction method is more susceptible to noise than the learning mechanism. However, we were curious about the influence of the changes made to the encoding scheme on the process of the construction and the quality of the resulting transformation vectors.

We expected the introduction of a type for each class of structures to solve the problem of ambiguous transformation results, because two input structures of different classes should no longer contain the identical role-filler bindings that caused the ambiguity when no types were used. On the other hand, the noise produced by the transformation might even further increase as the transformation process becomes more complex. The transformation vector now has to not only change the operations and operands of the structures but also to transform the type of an input structure into the correct type of the output structure. This increases the number of components in the transformation vector. The transformation vector transforming the class $x_1 \wedge \neg x_2$ into the class $\neg (x_1 \rightarrow x_2)$ is now constructed as

$$\hat{\mathbf{T}}_1 = \mathbf{type\_2}' \circledast \mathbf{type\_4} \circledast \mathbf{not} \circledast (\mathbf{conj}' \circledast \mathbf{impl} + \mathbf{first}' \circledast \mathbf{ante} + \mathbf{second}' \circledast \mathbf{not}' \circledast \mathbf{cons})$$

or

$$\begin{aligned}\hat{\mathbf{T}}_2 = {} & \mathbf{type\_2}' \circledast \mathbf{type\_4} \circledast \mathbf{not} \circledast \mathbf{conj}' \circledast \mathbf{impl} + \\ & \mathbf{type\_2}' \circledast \mathbf{type\_4} \circledast \mathbf{not} \circledast \mathbf{first}' \circledast \mathbf{ante} + \\ & \mathbf{type\_2}' \circledast \mathbf{type\_4} \circledast \mathbf{second}' \circledast \mathbf{cons} \ .\end{aligned}$$

The transformation vector for the transformation of the class $\neg x_1 \wedge \neg x_2$ into the class $\neg (x_1 \vee x_2)$ can be constructed as

$$\hat{\mathbf{T}}_3 = \mathbf{type\_1}' \circledast \mathbf{type\_3} \circledast \mathbf{not} \circledast (\mathbf{conj}' \circledast \mathbf{disj} + \mathbf{first}' \circledast \mathbf{not}' \circledast \mathbf{d\_first} + \mathbf{second}' \circledast \mathbf{not}' \circledast \mathbf{d\_second})$$

and can be simplified to

$$\begin{aligned}\hat{\mathbf{T}}_4 = {} & \mathbf{type\_1}' \circledast \mathbf{type\_3} \circledast \mathbf{not} \circledast \mathbf{conj}' \circledast \mathbf{disj} + \\ & \mathbf{type\_1}' \circledast \mathbf{type\_3} \circledast \mathbf{first}' \circledast \mathbf{d\_first} + \\ & \mathbf{type\_1}' \circledast \mathbf{type\_3} \circledast \mathbf{second}' \circledast \mathbf{d\_second} \ .\end{aligned}$$

The combined transformation vector is

$$\hat{\mathbf{T}}_{2+4} = (\mathbf{type\_2'} \circledast \mathbf{type\_4} \circledast \mathbf{not} \circledast \mathbf{conj'} \circledast \mathbf{impl} + \ldots) + (\mathbf{type\_1'} \circledast \mathbf{type\_3} \circledast \mathbf{not} \circledast \mathbf{conj'} \circledast \mathbf{disj} + \ldots) .$$

For simplicity we again only consider the parts of the structures and transformation vector regarding the operations. With the introduction of types, structures of the class $x_1 \wedge \neg x_2$ are represented by HRRs of the form $(\mathbf{type\_2} \circledast \mathbf{op} \circledast \mathbf{conj} + \ldots)$, whereas structures of the class $\neg x_1 \wedge \neg x_2$ are now differently represented by vectors of the form $(\mathbf{type\_1} \circledast \mathbf{op} \circledast \mathbf{conj} + \ldots)$. Convolving the former with $\hat{\mathbf{T}}_{2+4}$ results in

$$(\mathbf{type\_2} \circledast \mathbf{op} \circledast \mathbf{conj} + \ldots) \circledast \hat{\mathbf{T}}_{2+4} =$$

$$(\mathbf{type\_2} \circledast \mathbf{op} \circledast \mathbf{conj} + \ldots) \circledast (\mathbf{type\_2'} \circledast \mathbf{type\_4} \circledast \mathbf{not} \circledast \mathbf{conj'} \circledast \mathbf{impl} + \ldots + \mathbf{type\_1'} \circledast \mathbf{type\_3} \circledast \mathbf{not} \circledast \mathbf{conj'} \circledast \mathbf{disj} + \ldots) \approx$$

$$(\mathbf{type\_2} \circledast \mathbf{op} \circledast \mathbf{conj} \circledast \mathbf{type\_2'} \circledast \mathbf{type\_4} \circledast \mathbf{not} \circledast \mathbf{conj'} \circledast \mathbf{impl} + \ldots + \mathbf{type\_2} \circledast \mathbf{op} \circledast \mathbf{conj} \circledast \mathbf{type\_1'} \circledast \mathbf{type\_3} \circledast \mathbf{not} \circledast \mathbf{conj'} \circledast \mathbf{disj} + \ldots) \approx$$

$$(\mathbf{op} \circledast \mathbf{type\_4} \circledast \mathbf{not} \circledast \mathbf{impl} + \underbrace{\mathbf{type\_2} \circledast \mathbf{op} \circledast \mathbf{type\_1'} \circledast \mathbf{type\_3} \circledast \mathbf{not} \circledast \mathbf{disj}}_{noise} + \ldots).$$

In contrast to the result of the transformation without types, the second term of this expression is a cross term that does not represent a valid part of a structure in the domain and can therefore be treated as extra noise. The result of the transformation is

$$\approx \mathbf{type\_4} \circledast \mathbf{not} \circledast (\mathbf{op} \circledast \mathbf{impl} + \ldots) + \mathit{noise} .$$

This representation unambiguously describes a structure of the form $\neg(\ldots \rightarrow \ldots)$.

Convolving the superimposed transformation vector with a structure of the class $\neg x_1 \wedge \neg x_2$ now results in

$$(\mathbf{type\_1} \circledast \mathbf{op} \circledast \mathbf{conj} + \ldots) \circledast \hat{\mathbf{T}}_{2+4} =$$

$$(\mathbf{type\_1} \circledast \mathbf{op} \circledast \mathbf{conj} + \ldots) \circledast (\mathbf{type\_2'} \circledast \mathbf{type\_4} \circledast \mathbf{not} \circledast \mathbf{conj'} \circledast \mathbf{impl} + \ldots + \mathbf{type\_1'} \circledast \mathbf{type\_3} \circledast \mathbf{not} \circledast \mathbf{conj'} \circledast \mathbf{disj} + \ldots) \approx$$

$$(\mathbf{type\_1} \circledast \mathbf{op} \circledast \mathbf{conj} \circledast \mathbf{type\_2'} \circledast \mathbf{type\_4} \circledast \mathbf{not} \circledast \mathbf{conj'} \circledast \mathbf{impl} + \ldots + \mathbf{type\_1} \circledast \mathbf{op} \circledast \mathbf{conj} \circledast \mathbf{type\_1'} \circledast \mathbf{type\_3} \circledast \mathbf{not} \circledast \mathbf{conj'} \circledast \mathbf{disj} + \ldots) \approx$$

$$(\underbrace{\mathbf{type\_1} \circledast \mathbf{op} \circledast \mathbf{type\_2'} \circledast \mathbf{type\_4} \circledast \mathbf{not} \circledast \mathbf{impl}}_{noise} + \mathbf{op} \circledast \mathbf{type\_3} \circledast \mathbf{not} \circledast \mathbf{disj} + \ldots).$$

Again, in contrast to the result of the transformation without types, the first term of this expression is a cross term that does not represent a valid part of a structure in the domain. It is therefore treated as extra noise and the result of the transformation is

$$\approx \textbf{type\_3} \circledast \textbf{not} \circledast (\textbf{op} \circledast \textbf{disj} + \ldots) + \textit{noise}$$

which unambiguously describes a structure of the form $\neg(\ldots \vee \ldots)$. Note that the ambiguities caused by other constituents of structures without types are now resolved in the same way. Applying the constructed transformation vector should thus no longer result in the errors observed for structures without types.

To test this hypothesis we repeated the tests 6,7, and 8 now constructing transformation vectors for structures formed with types. The results are presented in Table 6.8. Whereas for test 6 the number of wrongly decoded elements was reduced by about one

Test	Transformations				Elements	Errors	
6	6.2	$x_1 \wedge \neg x_2$	$\Longleftrightarrow$	$\neg(x_1 \rightarrow x_2)$			
	6.4	$\neg x_1 \wedge \neg x_2$	$\Longleftrightarrow$	$\neg(x_1 \vee x_2)$	1200	77	(6.42%)
7	6.1	$(x_1 \rightarrow x_2) \wedge x_1$	$\Longrightarrow$	x_2			
	6.2	$x_1 \wedge \neg x_2$	$\Longleftrightarrow$	$\neg(x_1 \rightarrow x_2)$	1300	118	(9.08%)
	6.4	$\neg x_1 \wedge \neg x_2$	$\Longleftrightarrow$	$\neg(x_1 \vee x_2)$			
8	6.1	$(x_1 \rightarrow x_2) \wedge x_1$	$\Longrightarrow$	x_2			
	6.2	$x_1 \wedge \neg x_2$	$\Longleftrightarrow$	$\neg(x_1 \rightarrow x_2)$	1900	405	(21.32%)
	6.3	$\neg x_1 \vee \neg x_2$	$\Longleftrightarrow$	$\neg(x_1 \wedge x_2)$			
	6.4	$\neg x_1 \wedge \neg x_2$	$\Longleftrightarrow$	$\neg(x_1 \vee x_2)$			

Table 6.8: Results of combining the construction of up to 7 different transformations of structures containing class types. The combined transformations, the number of elements in the output structures, and the number of wrongly decoded elements are shown.

half from the tests without types (cf. Table 6.4), for test 7 the number of errors was of the same magnitude as before and for test 8 the number of errors increased to over 20%. In contrast to the previous results, however, the decoding errors were not concentrated on the implication and disjunction in structures of the classes $\neg(x_1 \rightarrow x_2)$ and $\neg(x_1 \vee x_2)$, but were distributed over different elements in all structures involved and

increased drastically with the number of transformations performed in parallel. These errors were no longer caused by identical role-filler bindings in the input structures of different classes. As in the process of learning the transformation vector from examples, using types solved the problem of similar representations for different classes of structures at the cost of increasing the noise in the transformation results.

As seen in the previous Section, the influence of the noise on the result structures can be reduced by increasing the size of the HRR vectors. However, constructing the transformation vector with class types (Table 6.8) caused considerably more errors than learning it (Table 6.6). We thus would not expect the constructing method to outperform the learning algorithm when the vector dimensionality is increased.

6.3 Summary

In this Chapter we used 13 different classes of structures to investigate the possibility of performing a number of holistic transformations of HRR structures in parallel. Earlier observations had shown that a transformation vector for a single transformation and its reverse can be learned from examples or constructed from the roles contained in the input and output structures of the transformation. The examples presented here demonstrate that under certain conditions these methods can also be applied to a number of transformations of different classes of structures. The two conditions that had to be met by the representations in our examples were

- sufficient dissimilarity and
- sufficiently large vector dimensionality.

The first condition enabled the transformation vector to distinguish between classes of structures that require different changes to their constituents in the course of the transformation. Sufficient dissimilarity of structures not belonging to the same class was achieved by convolving each structure with a special element representing the type of its class. The cluster analysis of seven input classes showed that introducing types separated clusters for different classes of structures which were previously overlapping, leading to a well structured representational space that allowed for the accurate parallel transformation of all classes in our example set.

The second condition guaranteed that the transformed structures were recognisable and could be decoded into their elements despite the noise introduced by the complex transformation. As shown in Chapter 3 and Chapter 4, an optimal transformation vector, i.e., a transformation vector which does not introduce any noise to the result of the transformation, is unique for any pair of structures. Increasing the amount of noise in a system can therefore not completely be avoided, if more than one structure pair is transformed or, like in the experiments here, even several different classes of structures are transformed in parallel. However, the results obtained for a learned transformation vector showed that the influence of the noise on the recognisability of the transformed structures could be kept very low by using very high-dimensional HRR vectors. Recall that if circular convolution and correlation are implemented by Fast Fourier Transforms, the computational effort for learning the transformation vector from examples grows only by $\mathrm{O}(n \log n)$ with the vector dimension n. The use of high-dimensional HRR vectors thus does not put large restrictions on the computability of the tasks.

For the examples investigated here, both methods of acquiring a transformation vector, constructing it from role vectors and learning it from example transformations, yielded a transformation vector that was able to perform the parallel transformation of a number of classes of structures distinguished by types. However, a comparison of the two methods showed that learning the transformation vector from examples seems to be the more reliable process. Constructing a transformation vector not only becomes a complicated and complex process as more classes of structures are added, the amount of noise introduced to the results of our example transformations was also considerably higher for a constructed than for a learned transformation vector. These findings support our earlier arguments that learning the transformation vector from examples is to be preferred over the construction method.

Although the results presented in this Chapter are promising, they have to be treated with caution as only the first step towards a general mechanism for the parallel transformation of structurally different hierarchical objects. A considerable amount of detailed empirical work and, in particular, theoretical background is required to prove the method to be general. One of the remaining problems, for example, is to find a suitable measure for the required dissimilarity of classes of structures. How different do the representations of two classes of structures have to be in order for them to

be successfully transformed by the same transformation vector? And how does the signal-to-noise ratio in the transformed structures vary with the number of transformations stored in the transformation vector and with the vector dimensionality used? Moreover, it would be interesting to see how the parallel transformation of HRR structures compares with the transformation of different RAAM structures demonstrated by Niklasson and Sharkey (1997). Answering these questions and developing a general mechanism for performing a number of different holistic processes in parallel will be, we believe, a fruitful area of future research.

Chapter 7

Conclusion

Despite the success of connectionist networks to model low-level cognitive tasks such as pattern completion and categorisation, Symbolists have long considered connectionist architectures unsuitable for high-level cognitive modelling, claiming that the lack of structured representations in connectionist networks prohibits the systematic structure-sensitive processing of hierarchical objects that is required in high-level cognitive tasks such as natural language processing. This thesis provided evidence against this claim.

Early connectionist systems were based on the assumption that the similarity-based representations formed in homogeneous neural networks and their learned mappings eliminate the necessity to explicitly represent symbols and rules. However, many Connectionists acknowledge today the need to represent systematic compositional structure in connectionist architectures. They agree that the strong assumptions of early Connectionism should be abandoned in favour of more flexible modular connectionist architectures and mechanisms that enable the connectionist representation of structured objects (e.g., Levy and Pollack (2001); van Gelder (1990); Touretzky and Hinton (1988)). Increasing the representational power of neural networks and combining it with robust connectionist learning mechanisms could lead to explanations for high-level cognitive processing not provided by traditional symbolic and early connectionist systems.

An inherent property of human cognition is the capacity for systematic structure-sensitive processing. Any model of cognition, be it symbolic or connectionist, thus has to provide mechanisms for the representation of structured objects and to implement

systematic processes over these representations that match the human generalisation capacity. Considerable progress has been made regarding the connectionist representation of structured objects, most importantly revealing that connectionist representations of structured objects do not have to be syntactically structured themselves in order to be processed in a systematic structure-sensitive manner. However, the degree of systematicity observed for connectionist structure-sensitive operations over such representations is still limited.

The central aim of this thesis has been to show that connectionist representational mechanisms based on the concept of reduced representation and on the functional composition of hierarchical structures can support structure-sensitive processes which show a degree of systematicity previously unseen in connectionist architectures. In order to achieve this goal we identified adequate connectionist representations of hierarchical objects, developed mechanisms that learn appropriate structure-sensitive processes over such representations, and demonstrated the high generalisation capacity of these processes.

Adequate connectionist representations of hierarchical structures have to be able to bind constituents of structures together and to superimpose individual bindings to compositional objects, thereby taking into account the restricted number of representational resources available in connectionist architectures. We argued that the concepts of distributed representation, functional compositionality, and reduced representation provide connectionist networks with the means to solve these problems, and discussed four representational schemes that are aimed specifically at the representation of hierarchical objects in a way suitable for connectionist processing: Tensor Product Representation (Smolensky, 1990), RAAM (Pollack, 1988), HRR (Plate, 1994), BSC (Kanerva, 1997). At the heart of these schemes are special binding operations for distributed representations. With the exception of Tensor Products, these binding operations implement a functional compositionality that does not require an increase in representational resources with the size of the represented objects. This in turn permits the recursive use of representations of bindings as parts of larger structures. Moreover, when provided with an external clean-up memory, these schemes can implement chunking, i.e., breaking large structures into pieces of manageable size, which allows to store and process representations of objects of potentially unlimited size (Plate, 1997b).

We regard HRRs and representations formed in the hidden layers of RAAMs as two of the most promising candidates for the connectionist representation of hierarchical structured objects. Despite their many similarities - the representation of role-filler bindings, the implementation of reduced representation, the preservation of the structural similarity of objects in their representations - the two representational schemes differ in a number of properties:

- Representations of hierarchical objects in RAAMs are formed from similarity-based representations of elements in a domain by a learning process. In contrast, HRRs of complex objects are constructed from randomly generated element representations.

- The fixed number of input slots in a RAAM provide only a fixed number of roles within represented structures and hence a predefined number of bindings which can be superimposed at once. The mechanism of composing HRRs, on the other hand, allows for a more flexible representation of hierarchical objects which is reflected in their more complex spatial structure.

- RAAMs show a relatively poor generalisation capacity for the encoding and decoding of previously unseen structured objects which makes the extension of a domain difficult. The addition of new elements to a domain might require the re-training of the RAAM. Since the encoding of HRR structures is an on-the-fly mechanism based on the construction of bindings from randomly generated representations, a domain can easily be extended by generating new role and filler vectors and simply applying the binding and superposition operation to them. No generalisation from previously constructed object representations is required.

Although these properties together with excellent generalisation and scaling capacities favour HRRs over RAAM representations, learning seems to us the more plausible process of arriving at a representation and is likely to make better use of the representational space than a construction method. It would be interesting to see whether the generalisation capacity of HRR increases even further when similarity-based element representations are used. High-dimensional vector representations that express the

similarity of represented objects can, for example, be learned using Latent Semantic Analysis (Landauer and Dumais, 1997) where representations of words in natural language are formed dependent on the context in which they appear. Recently, Paccanaro and Hinton (2001) introduced Linear Relational Embedding, a learning mechanism that acquires representations of concepts in form of distributed vectors and representations of binary relations between these concepts as matrices. The learning method was demonstrated to have a good generalisation capacity in the family tree problem (Hinton, 1986). However, a mechanism for learning similarity-based vector representations which meet the conditions of HRR on the distribution of their elements has yet to be found.

Although Tensor Product Representation, RAAM, HRR, and BSC are probably the most studied mechanisms for the connectionist representation of hierarchical structured objects, it is important to note that they are not the only ones. For example, Murdock (1982, 1993) and Metcalfe (1982, 1997) used the non-circular form of convolution and vector addition to represent structured objects in distributed memory models. Weber (1992) developed a binding operation called Term Coding Operation which is used in a connectionist unification system. This operation is based on circular convolution and HRRs with slightly different mathematical properties than the HRRs used in this thesis. More recently, Rachkovskij and Kussul (2001) proposed Context-Dependent Thinning, a connectionist binding mechanism on the basis of sparse binary vectors which, they argue, are biologically plausible and allow for a high storage capacity in associative networks. Despite their implementational differences, all these schemes can be viewed within the same framework involving the binding and the superposition of vectors in order to form representations for hierarchical structures (Plate, 1997a). We would thus expect these schemes to share at least some of the properties demonstrated in this thesis for RAAM and HRR such as the spatial structure of representations and the systematic relationship between reduced and full representations of objects.

The systematic relationship between reduced and full representations is what distinguishes connectionist reduced representations from symbolic pointer-based representations. Symbolic pointers, like reduced representations, are smaller than the full representations of the objects they point to and can be expanded into the full represen-

tations when necessary. Unlike reduced representations, however, they do not provide any information about the object they point to other than its location. In order to manipulate or compare such objects, we have to follow the pointer and manipulate their full representations which usually requires searching for components and decomposing structures into their elements. The systematic relationship between reduced and full connectionist representations on the other hand enables connectionist architectures to process representations of hierarchical structures holistically, i.e., structured objects can be compared and modified without their explicit decomposition. Thus, holistic operations provide efficient mechanisms for the manipulation of structured objects not generally found in symbolic systems.

We have demonstrated in this thesis that connectionist holistic operations can implement tasks such as unification and classification which are believed to play a central role in cognitive processing. All four representational schemes introduced in Chapter 2 support holistic operations to some degree. Holistic operations over RAAM representations can be learned from examples by perceptrons or three-layer backpropagation networks. In contrast, holistic operations over Tensor Products, HRRs, and BSCs have so far been constructed from the role and filler vectors present in the hierarchical structures and the binding and superposition operations used in the particular representational scheme. These construction methods have a number of limitations which become particularly apparent in our analysis of constructed vectors performing the holistic transformation of a number of different classes of HRR structures. Firstly, the roles and fillers of the structures involved in the holistic processing and their structural arrangement have to be known in advance in order to construct the vectors needed to perform the holistic transformation. Secondly, the construction process becomes more complicated with the increasing complexity of the holistic operation and the number and size of the structures involved. The complexity of the transformation task also affects the noise introduced by a transformation which, in our simulations, increased drastically with the number of components in the transformation vector. Most importantly, for some transformations, particularly if they required to leave parts of the input structures unchanged, no accurate transformation vector could be constructed at all. Finally, even if a transformation vector can be constructed which successfully transforms all HRR structures of a class, the error caused by its application, i.e., the noise intro-

duced during the transformation, is not guaranteed to be minimal. This also applies to the method of superimposing transformation vectors for individual transformations of BSCs (Kanerva, 1998).

We have presented two new methods for learning the holistic transformation of HRRs from examples that overcome these limitations. Both learning methods minimise the mean square error for the transformation of all training examples. The first method based on gradient descent can be straightforwardly implemented in a connectionist architecture. However, it converges only very slowly on the solution for the learning problem. The one-shot learning algorithm, on the other hand, acquires the transformation vector in a single pass through the training examples and can be very efficiently implemented using Fast Fourier Transforms.

It is difficult to directly compare a constructed and learned transformation vector with respect to the individual bindings they contain, because the extraction of a particular element from the learned transformation vector requires some knowledge about the element it is bound to. However, the normalised dot product can be used as a measure for their similarity. In our experiments regarding single transformation tasks the constructed transformation vectors were fairly good approximations of the learned ones and generalised to unseen input structures with approximately the same accuracy. However, differences between constructed and learned transformation vectors became apparent for transformations that require the modification of a number of different classes of structures. Most notably, the construction method was more susceptible to noise leading in the worst case to over 20% decoding errors when a number of transformations were performed in parallel. Learning the transformation vector that minimised the mean square error of these transformations yielded considerably better results.

A comparison of our new learning methods with connectionist networks that learn the holistic transformation of RAAMs shows that the two representational schemes support holistic operations in different ways and to different degrees. Perceptrons and three-layer backpropagation networks are able to learn relatively simple holistic operations over previously learned RAAM structures. However, they fail to perform complex holistic operations likely to be required in high-level cognitive processing if the encoding of the RAAM representations and the holistic operation are not learned

in parallel. The interleaved training of both the RAAM and the network performing the holistic operation optimises the representations of the structures with respect to the holistic operation. It decreases the decoding performance of the RAAM, however, which can prove problematic if the holistic operation results in new hierarchical structures which require decoding after the operation is performed. When hierarchical structures involved in holistic processes are represented by HRR, such an optimisation does not seem to be necessary. This was shown by a direct comparison of both methods performing Niklasson's (1993) transformation task. This task required a parallel training of the RAAM and the transformation network in order to achieve level 3 systematicity, but could be learned for HRR structures formed independently of the transformation process.

A second difference between learning the holistic transformation of RAAM and HRR structures is the information required in order to generalise the acquired knowledge to novel elements in the processed structures. For RAAM structures this generalisation is based on the similarity of the novel element to elements in the training set. However, HRR elements are represented by randomly generated vectors. The generalisation of a holistic transformation to novel elements in HRR structures is based on the position of the element in the structure, i.e., on the role it is bound to. This not only simplifies the representation of atomic elements and the extension of a domain by new objects, it also parallels observations, for example, in human language processing where generalisations to new words are not only based on their similarity to known words but also on their relation to other words and their position in a sentence.

The most significant difference between the mechanisms for learning the holistic transformation of HRRs and RAAM structures is that the former possess a higher generalisation capacity. We were able to show that the holistic transformation of HRRs can be generalised not only to compositional structures containing novel elements but to structures of higher complexity than all training examples, still containing novel elements. This capacity corresponds to the highest level of systematic generalisation proposed by both Hadley (1992) and Niklasson and van Gelder (1994a) and, to the best of our knowledge, has not yet been achieved by any other connectionist learning method. Furthermore, it directly contradicts Symbolists' claims that connectionist architectures cannot achieve the degree of systematicity required for modelling high-

level cognition.

Symbolists have based this claim largely on the apparent lack of structure in connectionist representations. Representations of objects are usually distributed over a number of processing units, i.e., objects are represented by apparently unstructured binary or real-valued vectors. However, we have shown that such connectionist representations possess a spatial structure. The constituent structure of a represented object is encoded in the position of its vector representation in the high-dimensional vector space. Although this spatial structure is fundamentally different from the constituent structure of symbolic representations, it still supports the structure-sensitive processing of the representations.

Both HRRs and RAAM representations cluster the representational space with respect to the syntactic structure of the represented object. Within each group of syntactically equivalent objects, clusters are formed with respect to the constituents of the objects that are last bound to the structures or, if the structured objects are viewed as trees, are closest to the root. A holistic operation can be successfully learned, if the clustering emerging from the process of forming the representations of structures complies with the categorisation of the structures required for the holistic processing.

We applied two different clustering methods to HRRs and representations learned by RAAMs. Both methods were able to cluster both kinds of representations with respect to the syntactic structure of the represented objects. However, in contrast to the constructivist GNG network, PCA did not provide any information about the subsequent clustering of HRRs. So far, it is unclear to us why these differences in the clustering methods were observed. Intuitively, we would expect the hierarchical nature of the data and the high dimensionality of the vectors to influence the performance of different clustering techniques. These hypotheses need further careful investigation, however.

With the robust generalisation capacities presented here, holistic operations of HRRs could provide useful mechanisms within a model of higher-level cognitive processes. Promising implementations of some aspects of such models have been presented, for example, by Plate (1997b, 1998a), Eliasmith (1997), and Eliasmith and Thagard (2001) who proposed models of analogical mapping based on the representation of episodes by HRR. We have argued that an essential step towards fully func-

tional and explanatory models of high-level cognition is the integration of a number of different operations into a single architecture. We presented first results regarding the acquisition of a number of different holistic transformations of HRR structures in a single transformation vector. Our simulations indicate that different classes of structures can in principle be transformed by the same transformation vector given an appropriate clustering of the representational space. Structures supposed to be processed in the same way by a holistic operation should be found in the same cluster. In contrast, structures supposed to be processed differently by the holistic operation, have to be sufficiently different, i.e., their representations should belong to well-separated clusters in the representational space. In our example, sufficient dissimilarity was achieved by introducing special elements distinguishing different classes of structures which resulted in well-separated clusters in the representational space. The increased amount of noise in the resultant structures could be dealt with by increasing the dimensionality of the element representations.

These results raise a whole set of further research questions. How different do representations of different classes of structures have to be in order to be processed correctly by the same transformation vector? And does adding further transformations require a re-training of the system? Moreover, the generalisation of transformation vectors to structures of higher complexity has so far only been tested on structures with up to two more levels of embedding. We would expect the representational capacity of HRRs to become an issue for more complex generalisations, and chunking will have to be used in order to correctly encode and decode the structures involved in the transformation process. Finally, our work regarding the holistic operations of connectionist representations concentrated on HRR and RAAM representations and it would be interesting to transfer the newly developed mechanisms to other representational schemes.

Using the example of transformational inference systems, we have argued that holistic transformations of HRRs and similar connectionist representations of structures, even those possessing level 5 systematicity and performed in parallel, still lack the computational power required to fully implement all aspects of high-level cognitive processing. Holistic transformations, as observed so far in connectionist networks, will have to be augmented with mechanisms that distinguish valid from invalid trans-

formations and provide meaningful default solutions when transformations are applied to invalid input structures. These mechanisms require a further level of systematicity which has yet to be clearly defined for, and met by, connectionist architectures.

We began this work by asking whether connectionist systems can account for high-level cognition. This questions still remains unanswered. We believe, however, that we have come one step closer to an answer by demonstrating that architectures which combine connectionist learning mechanisms with the expressive power of symbolic representations possess a degree of systematicity that is close to matching human performance.

Bibliography

Andrews, R., Diederich, J., and Tickle, A. (1995). A survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-based systems*, 8(6).

Bechtel, W. (1999). The case for connectionism. In Lycan, W. G., editor, *Mind and Cognition*, pages 153–170. Blackwell, 2nd edition.

Billingley, P. (1995). *Probability and Measure*. John Wiley & Sons, 3rd edition.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford.

Blank, D. S., Meeden, L. A., and Marshall, J. B. (1992). Exploring the symbolic/subsymbolic continuum: A case study of RAAM. In Dinsmore, J., editor, *The Symbolic and Connectionist Paradigms: Closing the Gap*, pages 113–148. Erlbaum, Hillsdale, NJ.

Bradley, P., Fayyad, U., and Reina, C. (1999). Scaling EM (expectation-maximization) clustering to large databases. Technical Report MSR-TR98-35, Microsoft Research, Seattle.

Chalmers, D. J. (1990). Syntactic transformations on distributed representations. *Connection Science*, 2:53–62.

Davis, P. J. (1979). *Circulant Matrices*. John Wiley & Sons, New York.

Dempster, N., Laird, A., and Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, 39:185–197.

Dolan, C. P. and Smolensky, P. (1989). Implementing a connectionist production system using tensor products. In Touretzky, D. S., Hinton, G. E., and Sejnowski, T. J., editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 265–272, San Mateo, CA. Morgan Kaufmann.

Dyer, M. G. (1991). Symbolic neuroengineering for natural language processing: A multilevel research approach. In Barnden, J. A. and Pollack, J. B., editors, *High-Level Connectionist Models*, volume 1 of *Advances in Connectionist and Neural Computation Theory,* Barnden, J. A., series editor, pages 32–86. Ablex, Norwood, NJ.

Eliasmith, C. (1997). Structure without symbols: Providing a distributed account of high-level cognition. Southern Society for Philosophy and Psychology Conference, Atlanta, GA.

Eliasmith, C. and Thagard, P. (2001). Integrating structure and meaning: A distributed model of analogical mapping. In press.

Elman, J. L. (1989). Representation and structure in connectionist models. Technical Report 8903, CRL, La Jolla, CA.

Elman, J. L. (1991). Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7:195–225.

Everitt, B. S. (1993). *Cluster Analysis*. Edward Arnold,London, 3rd edition.

Fodor, J. A. (1975). *The Language of Thought*. Harvard University Press, Cambridge, MA.

Fodor, J. A. (1999). Why there still has to be a language of thought. In Lycan, W. G., editor, *Mind and Cognition*, pages 199–211. Blackwell, 2nd edition.

Fodor, J. A. and McLaughlin, B. P. (1990). Connectionism and the problem of systematicity - Why Smolensky's solution doesn't work. *Cognition*, 35:184–204.

Fodor, J. F. and Pylyshyn, Z. W. (1988). Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28:73–193.

Fritzke, B. (1995). A growing neural gas algorithm learns topologies. In Tesauro, G., Touretzky, D. S., and Lean, T. K., editors, *Advances in Neural Information Processing Systems 7*. MIT-Press.

Fritzke, B. (1997). Some competitive learning methods. Manuscript. Institute for Neural Computation, Ruhr-Universität Bochum.

Fritzke, B. (1999). Growing self-organizing networks - history, status quo, and perspectives. In Oja, E. and Kaski, S., editors, *Kohonen Maps*, pages 131–144. Elsevier.

Gallant, S. I. (1988). Connectionist expert systems. *Communications of the ACM*, 31:152–169.

Giles, C. and Omlin, C. (1993). Extraction, insertion and refinement of rules in dynamically driven recurrent networks. *Connection Science*, 5(3-4):307–328.

Hadley, R. F. (1992). Compositionality and systematicity in connectionist language learning. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, pages 659–664.

Hadley, R. F. (1994). Systematicity revisited. *Mind and Language*, 9(4):431–444.

Hadley, R. F. (1999). Connectionism and novel combinations of skills: Implications for cognitive architecture. *Minds and Machines*, 9(2):197–221.

Hair, J. F., Anderson, R. E., Tatham, R. L., and Black, W. C. (1995). *Multivariate data analysis*. Prentice Hall, New Jersey, 4th edition.

Hammerton, J. (1998). Holistic computation: Reconstructing a muddled concept. *Connection Science*, 10(1):3–19.

Harnad, S. (1990). The symbol grounding problem. *Physica D*, 42:335–346.

Haykin, S. (1999). *Neural Networks. A Comprehensive Foundation*. Prentice Hall International, Inc.

Hebb, D. O. (1949). *The Organization of Behavior: A Neuropsychological Theory*. Wiley, New York.

Hecht-Nielsen, R. (1990). *Neurocomputing*. Addison-Wesley, Reading, MA.

Hinton, G. E. (1981). Implementing semantic networks in parallel hardware. In Hinton, G. E. and Anderson, J. A., editors, *Parallel Models of Associative Memory*, pages 161–187. Erlbaum, Hillsdale, NJ.

Hinton, G. E. (1986). Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 1–12, Hillsdale, NJ. Erlbaum.

Hinton, G. E. (1990). Mapping part-whole hierarchies into connectionist networks. *Artificial Intelligence*, 46:47–75.

Hinton, G. E., McClelland, J. L., and Rumelhart, D. E. (1986). Distributed representations. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pages 77–109. MIT Press, Cambridge, MA.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational properties. *Proceedings of the National Academy of Sciences of the U.S.A*, 81:3088–3092.

Jordan, M. I. (1989). Serial order: A parallel, distributed processing approach. In Elman, J. L. and Rumelhart, D. E., editors, *Advances in Connectionist Theory: Speech*. Erlbaum, Hillsdale, NJ.

Kanerva, P. (1997). Fully distributed representation. In *Real World Computing Symposium (RWC'97)*, pages 358–365.

Kanerva, P. (1998). Large patterns make great symbols: An example of learning from example. In *Neural Information Processing Systems Conference 1998 (NIPS98)*, Denver, Colorado, U.S.A.

Kaufman, L. and Rousseeuw, P. J. (1990). *Finding Groups in Data. An Introduction to Cluster Analysis*. John Wiley & Sons, Inc.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69.

Landauer, T. K. and Dumais, S. T. (1997). A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review*, 104(2):211–240.

LeCun, Y. (1985). Une procédure d'apprentissage pour réseau à seuil assymétrique. In *Cognitiva 85: A la Frontière de l'Intelligence Artificielle des Sciences de la Connaissance des Neurosciences*, pages 599–604, Paris. CESTA.

Legendre, G., Miyata, Y., and Smolensky, P. (1991). Distributed recursive structure processing. In *Advances in Neural Information Processing Systems 3*, pages 591–597. Morgan Kaufmann.

Levy, S. and Pollack, J. (2001). Infinite RAAM: A principled connectionist substrate for cognitive modeling. In *ICCM2001*. Lawrence Erlbaum Associates.

MacQueen, J. (1967). Some methods for classification and analysis of multi-variate observations. In LeCam, L. and Neyman, J., editors, *Proceedings of the Fifth Berkley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. Berkley: University of California Press.

Marcus, G., Brinkmann, U., Clahsen, H., Wiese, R., Woest, A., and Pinker, S. (1995). German Inflection: The Exception that Proves the Rule. *Cognitive Psychology*, 5:189–256.

Marcus, G. F. (1998). Rethinking eliminative connectionism. *Cognitive Psychology*, 37:243–282.

McClelland, J. L. and Kawamoto, A. H. (1986). Mechanisms of sentence processing: Assigning roles to constituents. In McClelland, J. L. and Rumelhart, D. E., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 2: Psychological and Biological Models*, pages 272–325. MIT Press, Cambridge, MA.

McCulloch, W. S. and Pitts, W. H. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133.

Metcalfe, J. (1982). A composite holographic associative recall model. *Psychological Review*, 89:627–661.

Metcalfe, J. (1997). Predicting syndromes of amnesia from a composite holographic associative recall/recognition model (Charm). *Memory*, 5:233–253.

Miikkulainen, R. and Dyer, M. G. (1991). Natural language processing with modular neural networks and distributed lexicon. *Cognitive Science*, 15:343–399.

Minsky, M. and Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge.

Mitchell, T. M. (1997). *Machine Learning*. McGraw Hill.

Murdock, B. B. (1982). A theory for the storage and retrieval of item and associative information. *Psychological Review*, 89:609–626.

Murdock, B. B. (1993). TODAM2: a model for the storage and retrieval of item, associative, and serial-order information. *Psychological Review*, 100:183–203.

Neumann, J. (1996). Untersuchungen zu dekompositionellen Verfahren der Regelextraktion aus künstlichen neuronalen Netzwerken. Diploma Thesis, Technical University Chemnitz-Zwickau.

Neumann, J. (2000a). Holistic Transformation of Holographic Reduced Representations. In *Foundations of Connectionist-Symbolic Integration, Workshop Notes*, 14th European Conference on Artificial Intelligence, Berlin.

Neumann, J. (2000b). Learning Holistic Transformation of HRR from Examples. In *Proceedings of the 4th International Conference on Knowledge-Based Intelligent Engineering Systems & Allied Technologies, KES'2000*, volume 2, pages 557–560.

Newell, A. and Simon, H. A. (1976). Computer science as empirical enquiry: Symbols and search. *Communications of the ACM, 19:113–126.*

Niklasson, L. (1999). Extended encoding/decoding of embedded structures using connectionist networks. In *Proceedings of the 9th International Conference on Artificial Neural Networks (ICANN99), Edinburgh, Scotland.*

Niklasson, L. and Bodén, M. (1997). Representing structure and structured representations in connectionist networks. In Browne, A., editor, *Current Perspectives on Neural Computing*. IOP Press.

Niklasson, L. and Sharkey, N. (1997). Systematicity and generalisation in connectionist compositional representations. In Dorffner, G., editor, *Neural Networks and a new 'AI'*, pages 217 – 232. Thomson Computer Press.

Niklasson, L. and van Gelder, T. (1994a). Can connectionist models exhibit non-classical structure sensitivity? In *Proceedings of the Sixteenth Annual Conference of the Cognitive Society 1994, Atlanta*, pages 664 – 669, Hillsdale, NJ. Lawrence Erlbaum Associates.

Niklasson, L. and van Gelder, T. (1994b). On being systematically connectionist. *Mind and Language*, 9(3):289–302.

Niklasson, L. F. (1993). Structure sensitivity in connectionist models. In *Proceedings of the Connectionist Models Summer School*, pages 162–169. Lawrence Erlbaum Associates.

Paccanaro, A. and Hinton, G. (2001). Learning distributed representations of concepts using linear relational embedding. *IEEE Transactions on Knowledge and Data Engineering. Special Section on Connectionist Models for Learning in Structured Domains.*, Vol. 13(2).

Phillips, S. (1998). Are feedforward and recurrent networks systematic? Analysis and implications for a connectionist cognitive architecture. *Connection Science*, 10(2):137–160.

Piaget, J. (1955). *The Child's Construction of Reality*. Routledge & Kegan Paul.

Pinker, S. and Prince, A. (1988). On language and connectionism: Analysis of a parallel distributed processing model of language acquisition. *Cognition*, 28:73–193.

Plate, T. (1994). *Distributed Representations and Nested Compositional Structure*. PhD thesis, Department of Computer Science, University of Toronto.

Plate, T. (1997a). A common framework for distributed representation schemes for compositional structure. In Maire, F., Hayward, R., and Diederich, J., editors, *Connectionist Systems for Knowledge Representation and Deduction*, pages 15–34.

Plate, T. (1997b). Structure matching and transformation with distributed representations. In Sun, R. and Alexandre, F., editors, *Connectionist-Symbolic Integration*. Lawrence Erlbaum Associates.

Plate, T. (1998a). Analogy retrieval and processing with distributed represenations. Technical Report CS-TR-98-4, Victoria University of Wellington, Computer Science.

Plate, T. (1998b). *vcalc0.9*. A fast vector calculator for Holographic Reduced Representations. URL: http://pws.prserv.net/tap/#software.

Plunkett, K. and Marchman, V. (1991). U-shaped learning and frequency effects in a multi-layered perceptron: Implications for child language acquisition. *Cognition*, 38:43–102.

Pollack, J. B. (1988). Recursive auto-associative memory: Devising compositional distributed representations. In *Proceedings of the 10th Annual Conference of the Cognitive Science Society*, pages 33–39, Hillsdale, NJ. Erlbaum.

Pollack, J. B. (1989). Implications of recursive distributed representations. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 1*, pages 527–536. Morgan Kaufmann, San Mateo, CA.

Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence*, 46:77–105.

Rachkovskij, D. A. and Kussul, E. M. (2001). Binding and normalization of binary sparse distributed representations by context-dependent thinning. *Neural Computation*, 13(2):411–452.

Roche, E. and Schabes, Y., editors (1997). *Finite-State Language Processing*. MIT Press, Cambridge, MA.

Rojas, R. (1996). *Neural Networks. A Systematic Introduction.* Springer, Berlin.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408.

Rumelhart, D. E. and McClelland, J. L. (1986). On learning past tenses of English verbs. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 2: Psychological and Biological Models*, pages 216–271. MIT Press, Cambridge, MA.

Rumelhart, D. E., McClelland, J. L., and the PDP Research Group (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations.* MIT Press, Cambridge, MA.

Scott, D. W. (1992). *Multivariate Density Estimation: Theory, Practice, and Visualization.* John Wiley and Sons, New York.

Sejnowski, T. J. and Rosenberg, C. R. (1987). Parallel networks that learn to pronounce English text. *Complex Systems*, 1:145–168.

Shastri, L. and Ajjanagadde, V. (1993). From simple associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings using temporal synchrony. *Behavioral and Brain Sciences*, 16:417–494.

Silverman, B. W. (1985). *Density Estimation for Statistics and Data Analysis.* Chapman and Hall, London.

Smolensky, P. (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence*, 46:159–216.

Sperduti, A. (1994). Labeling RAAM. *Connection Science*, 6(4):429–459.

Sperduti, A., Starita, A., and Goller, C. (1995). Learning distributed representations for the classification of terms. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 509–515.

St. John, M. F. and McClelland, J. L. (1989). Applying contextual constraints in sentence comprehension. In Touretzky, D. S., Hinton, G. E., and Sejnowski, T. J.,

editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 338–346, San Mateo, CA. Morgan Kaufmann.

Stolcke, A. and Wu, D. (1992). Tree matching with recursive distributed representations. Technical Report TR-92-025, International Computer Science Institute, Berkeley, CA.

Sun, R. (1995). Robust reasoning: integrating rule-based and similarity-based reasoning. *Artificial Intelligence*, 75(2):241–296.

Sun, R. and Alexandre, F., editors (1997). *Connectionist-Symbolic Integration*. Lawrence Erlbaum Associates.

Touretzky, D. S. and Hinton, G. E. (1988). A distributed connectionist production system. *Cognitive Science*, 12:423–466.

Touretzky, D. S. and Pomerleau, D. A. (1994). Reconstructing physical symbol systems. *Cognitive Science*, 18(2):345–353.

Ueda, N., Nakano, R., Gharhamani, Z., and Hinton, G. (2000). SMEM algorithm for mixture models. *Neural Computation*, 12(9):2109–2128.

van Gelder, T. (1990). Compositionality: A connectionist variation on a classical theme. *Cognitive Science*, 14:355–384.

von Foerster, H. (1973). On constructing a reality. In Preiser, F. E., editor, *Environmental Design Research*, volume 2, pages 35–46. Dowden, Hutchinson & Ross.

von Glasersfeld, E. (1984). An introduction to radical constructivism. In Watzlawick, P., editor, *The Invented Reality*. Norton, New York.

Weber, V. (1992). Connectionist unification with a distributed representation. In *Proceedings of the International Joint Conference on Neural Networks – IJCNN '92, Beijing, China*, pages 555–560. IEEE.

Wermter, S. and Sun, R., editors (2000). *Hybrid Neural Systems*. Springer, Heidelberg.

Westermann, G. (2000). *Constructivist Neural Network Models of Cognitive Development*. PhD thesis, Division of Informatics, University of Edinburgh.

Willshaw, D. (1971). *Models of distributed associative memory*. PhD thesis, University of Edinburgh.

Willshaw, D. J., Buneman, O. P., and Longuet-Higgins, H. C. (1969). Non-holographic associative memory. *Nature*, 222:960–962.

Willshaw, D. J. and von der Malsburg, C. (1976). How patterned neural connections can be set up by self-organization. *Proceedings of the Royal Society of London, Series B*, 194:431–445.